



UNIVERSIDAD DE MÁLAGA



E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

INGENIERÍA INFORMÁTICA

Comparación de rendimiento entre bases de datos Relacionales, NoSQL y Blockchain

Comparación de rendimiento entre PostgreSQL, MongoDB y Kaleido

Performance comparison among Relational, NoSQL databases and Blockchain

Performance comparison among PostgreSQL, MongoDB and Kaleido

Realizado por
Alberto Pérez Román

Tutorizado por
Enrique Soler Castillo

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, febrero 2020



UNIVERSIDAD
DE MÁLAGA



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA INFORMÁTICA

**Comparación de rendimiento entre bases de datos
Relacionales, NoSQL y Blockchain**

Comparación de rendimiento entre PostgreSQL, MongoDB y Kaleido

**Perfomance comparison among Relational, NoSQL
databases and Blockchain**

Perfomance comparison among PostgreSQL, MongoDB and Kaleido

Realizado por
Alberto Pérez Román

Tutorizado por
Enrique Soler Castillo

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, FEBRERO 2020

Fecha defensa: febrero 2020



UNIVERSIDAD
DE MÁLAGA

Resumen

La correcta elección de una base de datos es fundamental a la hora de implementar una aplicación. La información necesita ser guardada y consultada, y es por ello por lo que se debe elegir una base de datos que mejor se adapte a las necesidades del proyecto que se desee elaborar, para evitar así tiempos de respuesta altos que puedan ocasionar un rendimiento malo a la hora de interactuar con la aplicación.

Otro de los motivos por los que se selecciona una base de datos es por su seguridad y es que hoy en día, se están empezando a poner de moda unas bases de datos que usan un tipo de tecnología muy característico que aportan una mayor seguridad a los datos, que es el denominado Blockchain. A raíz de esto, es por lo que nos hemos enfocado a llevar a cabo nuestro TFG, ya que queríamos ver cómo de buenas pueden llegar a ser y si realmente merecen la pena.

En este TFG, partimos de una en concreto, sobre el cual se va a construir 3 aplicaciones que usan diferentes tipos de formas de almacenar datos, con el fin de medir los tiempos de ejecución de las respuestas que obtengamos de su uso. Finalmente, compararemos los tiempos para así determinar cuál de las 3 opciones elegidas ofrece mejor rendimiento y cuál tiene mayor grado de dificultad.

Estas aplicaciones usarán una base de datos relacional, una no relacional y una que usa una tecnología novedosa conocida como Blockchain. En concreto las opciones son PostgreSQL, MongoDB y Kaleido, respectivamente.

Palabras clave: PostgreSQL, MongoDB, Kaleido, bases de datos Relacionales, bases de datos NoSQL, Blockchain, comparativa de rendimiento.

Abstract

The correct choice of a database is essential when we are implementing an application. The information needs to be saved and consulted, and that is why you should choose a database that best suits the needs of the project you want to prepare, to avoid high response times that can cause poor performance.

Another choice's reason is its security because nowadays, they are starting to put in fashion databases that use a very characteristic type of technology that provide greater data security, which is the so-called blockchain. As a result, this is why we have focused on carrying out our TFG, since we wanted to see how good they can be and if they really are worth it.

In this TFG, we start from a specific situation, building 3 applications that use different types of data, in order to measure the execution times of the responses they obtain from their use. Finally, we will compare the times to determine which technology gives the best performance and which has the biggest difficulty level.

These applications will use a relational database, a non-relational database and a new technology known as Blockchain. Specifically, the options are PostgreSQL, MongoDB and Kaleido, respectively.

Keywords: PostgreSQL, MongoDB, Kaleido, Relational databases, NoSQL databases, Blockchain, performance comparison.

Índice

Resumen	1
Abstract	1
Índice	1
Introducción	1
1.1 Motivación	1
1.2 ¿Qué son las bases de datos?	2
1.3 Clasificación	2
1.3.1 Variabilidad.....	3
1.3.2 Contenido	4
Modelos de BBDD	5
2.1 Relacionales.....	6
2.2 No relacionales	8
2.3 Blockchain	11
Análisis y organización.....	15
3.1 Objetivos	16
3.2 Metodología usada	16
3.3 Requisitos funcionales.....	17
3.4 Requisitos no funcionales.....	18
3.5 Casos de uso	19
3.6 Diagrama de caso de uso	23
Diseño e implementación	25
4.1 PostgreSQL	25
4.1.1 Descripción	25
4.1.2 Modelo	26
4.1.3 Implementación.....	27
4.2 MongoDB	30
4.2.1 Descripción	30
4.2.2 Modelo	31
4.2.3 Implementación.....	31
4.3 Kaleido	35
4.3.1 Descripción	35
4.3.2 Modelo	35
4.3.3 Implementación.....	36
Manual de Usuario	41
Resultados	49
6.1 Pruebas de rendimiento	49
6.1.1 PostgreSQL.....	49
6.1.2 MongoDB.....	51

6.1.3 Kaleido	52
6.2 Dificultades.....	54
Conclusiones	55
7.1 Conclusión	55
7.2 Opinión sobre el trabajo.....	57
Bibliografía.....	59

1

Introducción

1.1 Motivación

Seleccionar correctamente qué base de datos usar es una elección que se debe de hacer cuidadosamente, siempre teniendo en cuenta varios factores como son el costo, el tiempo y su funcionalidad, y es por esto por lo que sea trate de una de las decisiones determinantes para tener en cuenta ya que puede influir en el éxito de un proyecto.

Cabe decir que todas las bases de datos son diferentes, y pese a estar clasificadas dentro de un mismo tipo, internamente funcionan de forma distinta ya que su arquitectura no es la misma. Hay que tener muy claro la cantidad de datos a tratar, los usuarios que van a estar ejecutando la aplicación simultáneamente, si el entorno de administración es intuitivo y de fácil aprendizaje, su escalabilidad, ya que progresivamente irá aumentando el número de registros y hay que saber que hacer. En definitiva, escoger la base de datos que mejor se ajuste a tus requerimientos.

La manera tradicional de asegurar la persistencia de los datos es mediante la utilización de bases de datos relaciones. Sin embargo, para afrontar nuevos desafíos como la gestión documental o el almacenamiento de datos masivos como los generados por dispositivos IOT, han surgido nuevos tipos de bases de datos como las no relacionales (NoSQL) y, últimamente, están cobrando bastante importancia las soluciones basadas en el almacenamiento distribuido de la información utilizando la tecnología Blockchain.

Actualmente, existen muchas formas de bases de datos, ya sea desde una biblioteca, hasta los enormes conjuntos de datos de usuario que tienen las grandes empresas de telecomunicaciones.

El enfoque de nuestro TFG va ligado a este último tipo de soluciones, las bases de datos que usan tecnología Blockchain, ya que este tipo de bases de datos están cobrando verdadera importancia en la actualidad y cada día más empresas están empezando a invertir en soluciones con este tipo de tecnología.

1.2 ¿Qué son las bases de datos?

El término base de datos, es un concepto bastante usado hoy en día no solo en las empresas, que requieren de su uso para guardar información relevante, sino también para uso propio con el fin de asegurar tus datos, para bibliotecas públicas donde se almacenen registros de los libros, etc.

Se entiende por bases de datos a un conjunto de información almacenada de forma ordenada, perteneciente a un mismo contexto y que presenta una estructura específica con el fin de que su contenido pueda ser posteriormente tratado y analizado de la forma más rápida y sencilla posible.

Las bases de datos surgen ante la necesidad de almacenar información, debido a que se pretendía hallar una forma para que los datos pudieran ser conservados con el paso del tiempo sin llegar a deteriorarse permitiendo, además, que esa información almacenada pudiese ser consultada ya que, de lo contrario, no tendría mucho sentido.

Es por ello por lo que están diseñadas de tal modo que el usuario pueda encontrar los datos que busca con facilidad. La información de estos datos se encuentra almacenada siguiendo una estructura dictada por distintos parámetros. Esto se lleva a cabo mediante sistemas de gestión de bases de datos que, de forma automatizada, permiten que los datos sean almacenados de forma ordenada y que estos sean recuperados.

A la hora de crear una base de datos, existen muchos tipos distintos de modelos, cada uno con sus propias características: mejor seguridad, buena estructuración de los datos, fácil aprendizaje, buenos tiempos de respuesta, problemas de compatibilidad con otros programas... En definitiva, se hace uso de modelos de bases de datos que, en función de tus necesidades, permiten el diseño y la implementación de algoritmos y otros mecanismos lógicos para gestionar los datos.

1.3 Clasificación

A la hora de clasificar las bases de datos, estos se clasifican en función de su estructura, su uso, su contexto y las necesidades que busques. Existen dos formas principales de clasificación que se centran en la variabilidad de los datos y en el contenido a analizar. También se agrupan en función de los modelos de administración de bases de datos, pero esto lo abarcaremos en profundidad más adelante. [1]

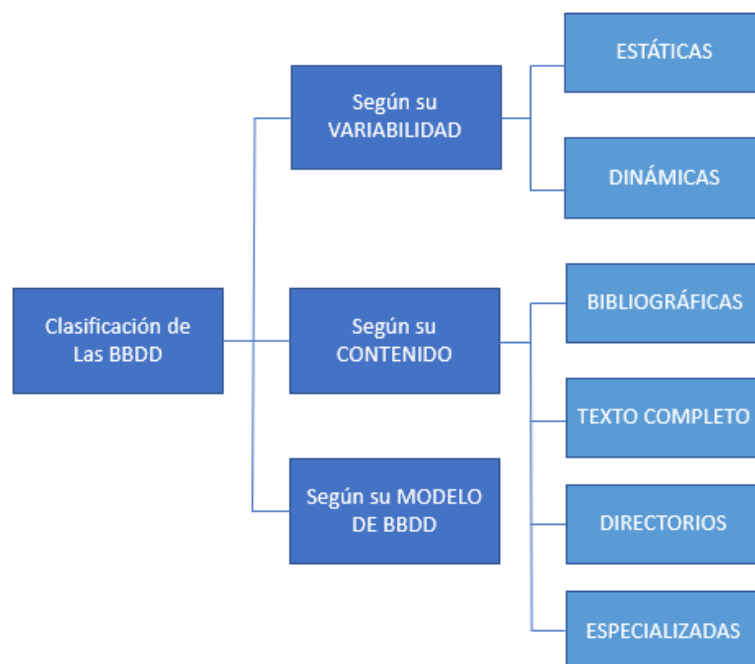


Figura 1.1: Clasificación de las bases de datos en función de su variabilidad y su contenido, junto a una tercera clasificación (según el modelo).

1.3.1 Variabilidad

Esta primera clasificación va enfocada, como su propio nombre indica, a la variabilidad de la base de datos, es decir, en cómo se estructuran los datos dentro de la base de datos. Se tratan de las bases de datos estáticas y las bases de datos dinámicas. [2]

Las bases de datos **estáticas** son bases de datos centradas en la lectura. Esto se ve reflejado en sus datos, ya que no van a ser modificados. Son utilizadas generalmente para almacenar datos históricos que sean relevantes con el fin de, posteriormente, poder realizar estudios para comparar su relevancia con respecto al tiempo. Ejemplos de este tipo de bases de datos serían periódicos o bibliotecas donde la información se almacena, para consultarla en un futuro.

Al contrario que las estáticas, las bases de datos **dinámicas** son bases de datos en las que a menudo se realizan inserciones de datos y eliminando otros, están en continuo cambio. El hecho de que puedan ser modificadas con el paso del tiempo, permite al propietario tener actualizados todos los registros. El ejemplo más claro se encuentra en las tiendas, donde su inventario se va modificando, y si se acaba un producto, este se renueva, o se incluyen otros más novedosos.

En definitiva, la principal diferencia entre estos dos tipos de bases de datos es que las estáticas se centran en la lectura de datos mientras que, en las dinámicas, podemos consultar y modificar la información.

Esto resulta realmente interesante para las empresas ya que, por ejemplo, no hace faltan invertir muchos recursos en una base de datos dinámica, si luego solo se va a usar para añadir y leer datos en ella, con el fin de tomar decisiones que influyan al futuro de la empresa.

1.3.2 Contenido

Esta segunda clasificación se centra en el contenido a analizar. Dentro de esta categoría se encuentran las bases de datos bibliográficas, las de texto completo, los directorios y las especializadas. [3]

Las bases de datos **bibliográficas** son bases de datos donde se guardan registros con información clasificada. Puedes hallar información sobre la persona que lo ha publicado tales como el autor, el título o la fecha de la publicación. Una cosa a tener en cuenta sobre este tipo de bases de datos es que no se pueden conseguir la totalidad de información del documento. Un ejemplo puede ser una colección de información resultante del análisis de laboratorio.

Las bases de datos de **texto completo** son bases de datos muy parecidas a las bibliográficas, ya que permiten buscar términos específicos, palabras clave, con la diferencia de que en este tipo de bases de datos se pueden consultar la información del documento al completo que se encuentra almacenado. Un ejemplo son el todo el contenido de todas las ediciones de una colección de revistas científicas.

Por otro lado, se encuentran los **directorios**, que son bases de datos con las que tratamos a diario y suelen ser utilizadas con fines empresariales, de ahí que exista una subcategoría dentro de ellas, las personales y las empresariales. Un ejemplo son las guías telefónicas en formato electrónico.

Dentro de las empresariales, se dividen en función de su información: si contienen nombres y direcciones, si contiene contactos telefónicos y direcciones de correo electrónico y, por último, si contiene datos relacionados con la facturación, los códigos postales o números de empleado.

Con respecto a los directorios personales únicamente hay un tipo, ya que hay leyes que se encargan de proteger la privacidad de los usuarios pertenecientes al directorio, como es el caso de la Ley Orgánica de Protección de Datos (LOPD) en España.

Por último, existen otro tipo de bases de datos, las **especializadas**, creadas a partir de unas ciertas necesidades con el fin de almacenar información que vaya a ser consumida por un determinado público. Se tratan de bases de datos con información muy específica y técnica.

2

Modelos de BBDD

A continuación, vamos a hablar sobre las clasificaciones de bases de datos centradas en los distintos modelos de administración de datos que utilizan. Algunos tipos de bases de datos siguen una estructura acorde a la forma en la que guardan los datos, mientras que otras dependen de los métodos de almacenamiento y recuperación de estos.

Dentro de esta clasificación, se encuentran las bases de datos **jerárquicas**. En este tipo de bases de datos, la información se almacena en una estructura jerárquica, donde los datos están organizados en una especie de árbol invertido. Está compuesto por nodos (padre, hijo y raíz) y ramas que conectan dichos nodos. Este tipo de bases de datos no son sencillas de construir y son complejas de modificar una vez diseñada, sin embargo, permiten la globalización e independencia de los datos, así como la integridad y la posibilidad de compartir la información entre los usuarios gracias a como se encuentra estructuradas.

Frente a las jerárquicas, surgieron otro tipo de bases de datos que representaron un gran avance. Se trata de las bases de datos de **red**, que son muy parecidas a las jerárquicas teniendo como diferencia principal la composición del nodo ya que, en este caso, pueden tener varios padres. Esto permitió solucionar los problemas de redundancia de datos que poseía el modelo jerárquico.

Este tipo de bases de datos no se suelen usar en la actualidad debido a su grado de complejidad a la hora de diseñarla y al ser difíciles de modificar, al igual que pasa con las jerárquicas.

Existen más tipos distintos, como las bases de datos **transaccionales** (enfocadas sobre todo al envío y recepción de datos a gran velocidad), las bases de datos **multidimensionales** (muy parecidas a las relacionales) o las bases de datos orientadas a objetos (se almacena el objeto completo, no información sobre él).

Pero nosotros en este TFG nos vamos a centrar en las bases de datos relacionales, las bases de datos no relacionales y las bases de datos que usan la tecnología Blockchain que explicaremos a continuación.

2.1 Relacionales

Las bases de datos relacionales han sido las más utilizadas durante una gran cantidad de tiempo, cuyo núcleo se focaliza en el uso de relaciones entre datos. La información es recuperada y almacenada mediante consultas, construidas habitualmente en SQL (Structured Query Language). Se trata de un estándar implementado por los sistemas de gestión de bases de datos relacionales más conocidos. [4]

El funcionamiento de este tipo de bases de datos consiste en introducir los datos en registros, que se encuentran almacenados en tablas. Gracias a estas bases de datos, se pueden relacionar de forma sencilla los elementos entre sí y establecer conexiones entre ellas, formando así relaciones entre los registros.

El modelo de datos que permite representar las entidades de la base de datos se denomina modelo entidad-relación (E/R). Para ello se elabora un diagrama entidad-relación y posteriormente, se le añaden atributos y una descripción de restricciones que no aparezcan en el diagrama.

Las entidades son objetos del mundo real como puede ser una persona, un animal, un vehículo... A estas entidades, se le asignan unas características, denominadas atributos, que básicamente, sirven para definir e identificar a la entidad y existen muchos tipos de atributo.

Por ejemplo, supongamos que tenemos una entidad Persona, pues esa entidad tendrá atributos tales como Nombre, Edad. Pues dentro de los atributos de la entidad, cada registro puede tener valores distintos como, por ejemplo: Alberto, 23 y Rafa, 23. LA cuestión es que hay que designar un atributo que sirva de identificador y que no se repita para poder diferenciar los registros ya que, si por un casual existen dos personas con el mismo nombre y edad, no se podrían diferenciar. Para ello se designaría como clave primaria DNI, un identificador único que nos permita identificar a las distintas personas, aunque tengan el resto de los atributos con el mismo valor. Si por algún casual se desconoce el valor de cierto atributo, este se designa a nulo, que es un tipo especial de valor usado en informática.

Sobre las entidades y las relaciones, existen restricciones, que son reglas que se deben respetar y se da entre dos entidades. Las restricciones pueden ser de uno a uno (1:1), de uno a muchos (1:N) y de muchos a muchos (N:N).

Trasladando este concepto sobre nuestro caso de uso que más adelante analizaremos, la restricción de 1:1 se da entre Animal y Sacrificio, ya que el animal solo puede ser sacrificado una vez; la restricción de 1:N se da entre Animal e Inspección, ya que un animal puede recibir muchas inspecciones; y la de N:N se da entre Animal y Transporte, porque en un transporte pueden haber sido transportado varios animales y porque un animal puede haber sido transportado varias veces.

Dentro de los atributos, como hemos hablado anteriormente, existen identificadores denominado claves, que se denominan de una forma distinta dependiendo de la función que desempeñen. Están las claves primarias (claves únicas formada por uno o más campos de la tabla que define de forma única a los demás datos de la tabla y se usa para establecer relaciones), las claves foráneas (referencia a una clave de otra tabla que determina la relación entre dos tablas) y la clave índice (usadas para tener un acceso más rápido a los datos).

En el diseño, existe un proceso conocido como normalización, que consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo E/R al modelo relacional con el fin de reducir la redundancia de datos.

Cabe destacar, que todas las transacciones de la base de datos deben ser atómicas, coherentes, aisladas y duraderas (ACID) para garantizar la integridad de los datos. El ser atómico significa que la transacción completa se ejecute correctamente (si una parte de ella falla, toda la transacción queda descartada). El significado de la coherencia hace referencia a los datos de la base de datos, ya que estos deben cumplir las reglas establecidas, restricciones, disparadores y cascadas.

Por otro lado, el aislamiento es vital para la concurrencia y ayuda a asegurar que las transacciones sean independientes entre sí. Finalmente, el hecho de ser duraderas requiere que una vez se haya completado la transacción, los cambios que se hayan hecho deben ser permanentes.

Las principales ventajas de las bases de datos relacionales son que garantizan la integridad referencial (al eliminar un registro, elimina los registros que dependían de este a través de relaciones) y que no haya duplicidad en los registros (gracias al uso de claves primarias y de claves únicas), además de favorecer la normalización haciéndola más comprensible.

Por otro lado, las desventajas son que, en lo relacionado a datos gráficos o multimedia, presenta deficiencias y con respecto a los bloques de texto, estos no se manipulan muy bien como tipo de dato.

Entre los gestores más populares encontramos: MySQL, PostgreSQL, Oracle, Microsoft SQL Server, DB2. Uno de estos gestores, PostgreSQL, es la elección dentro de la categoría de bases de datos relacionales, que vamos a explicar, usar y comparar posteriormente con el fin de determinar si este gestor es el que mejor rendimiento otorga para las pruebas que realizaremos.



Figura 2.1: Gestores de bases de datos relacionales más usados en la actualidad.

2.2 No relacionales

Este tipo de bases de datos están diseñadas para modelos de datos muy específicos y que hace uso de esquemas flexibles con el fin de crear aplicaciones modernas. También se le conoce por el nombre de bases de datos NoSQL.

Son bases de datos bastante conocidas debido a su facilidad sobre todo con respecto a su desarrollo, tienen una gran funcionalidad y un gran rendimiento. Dentro de ellas, cada una sigue un modelo de datos distinto, como pueden de documentos, de gráficos, de clave-valor, en memoria y de búsqueda que abarcaremos más adelante.

Surgen como alternativa a las bases de datos relacionales, diferenciándose de estas en que las no relaciones no siguen un modelo relacional y usan una forma de almacenamiento estructurada, es decir, sus tablas no poseen una estructura fija. Suelen ser, por tanto, bases de datos muchos más flexibles y abiertas. Esto permite la adaptación de este tipo de bases de datos a las necesidades específicas de los proyectos de una forma más sencilla que los modelos E/R.

Este tipo de bases de datos, debido a su optimización, son recomendables para aplicaciones que necesitan grandes volúmenes de datos, baja latencia y modelos de datos flexibles. Se suelen usar en entornos distribuidos que han de estar siempre operativos y disponibles.

Con respecto a su funcionamiento, en una base de datos NoSQL tomando como referencia el ejemplo de las relacionales, el registro de un objeto Persona se almacena como un documento JSON. Este tipo de documentos son un formato de texto sencillo usado para el intercambio de datos que proviene del acrónimo JavaScript Object Notation.

Para cada Persona, los elementos nombre y edad se almacenan como atributos de un solo documento. De este modo, los datos se encuentran optimizados permitiendo un desarrollo intuitivo y una escalabilidad horizontal.

```
{
  "persona": {
    "nombre": "Alberto",
    "edad": "23"
  }
}
```

Figura 2.2: Estructura de ejemplo que refleja cómo estaría construido un documento JSON.

Dentro de las bases de datos no relaciones, existen 5 tipos, en función de los modelos de datos que sigan. En un primer lugar se encuentran las bases de datos **clave-valor**, que poseen alta divisibilidad y permiten escalado horizontal a grandes escalas. Este modelo de datos se ajusta a casos de uso como juegos, tecnología publicitaria e IoT (Internet of Things).

En las bases de datos de **documentos**, los datos se representan como un objeto o un documento JSON. Estas proporcionan facilidades a la hora de almacenar y consultar datos, y permiten que evolucionen según las necesidades de cada uno gracias a su naturaleza flexible y semiestructurada. Suele ser usado en catálogos o en sistemas de administración de contenido que poseen documentos únicos que evolucionan con el paso del tiempo.

Luego se encontrarían las bases de datos **en memoria**, donde sus datos se encuentran almacenados en la memoria principal haciendo más fácil la obtención de tiempos de respuesta más rápidos y son muy útiles para procesos de consulta. Suele darse en aplicaciones de juegos que necesitan tiempos de sesión y análisis en tiempo real.

Por otro lado, se encuentran las bases de datos de **gráficos** que se centran en facilitar la creación y la ejecución de aplicaciones que funcionan con datos que se encuentran altamente conectados. Suelen darse en redes sociales, motores de recomendaciones, entre otros.

Por último, se encuentra las bases de datos **de búsqueda**. Este tipo de bases de datos se dan en aplicaciones que generan registros con el fin de ayudar a los desarrolladores a solucionar problemas. Para este tipo de bases de datos se usa un poderoso motor de búsqueda de alto rendimiento como es el caso de Amazon ES (Amazon Elasticsearch Service) diseñado con el fin de proporcionar visualizaciones en tiempo real y análisis de datos generados por máquinas al indexar, agregar y buscar métricas semiestructuradas.
[5]

Las ventajas de este tipo de bases de datos residen en su flexibilidad (ideales para datos semiestructurados y no estructurados otorgando un desarrollo más rápido e intuitivo), en su escalabilidad (permitiendo aumentar sus capacidades mediante clústeres distribuidos de hardware), en su alto rendimiento (gran optimización hacia los modelos de datos específicos y patrones de acceso) y en su alta funcionalidad (posee tipos de datos que se ajustan específicamente a modelos de datos concretos).

Como desventajas cabe destacar que sufren problemas de compatibilidad al no estar suficiente maduros para algunas empresas ya que, al ser relativamente novedosas, puede que no ofrezcan mucha documentación y las herramientas que se usan para administrar suele ser por consola de comando careciendo de una interfaz de usuario intuitiva. También cabe decir que cómo existen diversos tipos, no hay un estándar definido entre los distintos motores y en algunos tipos no poseen información atómica.
[6]

Entre las bases de datos NoSQL más conocidas podemos destacar MongoDB (orientado a ficheros), Redis (basado en el almacenamiento clave-valor), Cassandra (basado también en el almacenamiento clave-valor) o CouchDB (la información se almacena en documentos JSON).

En concreto, MongoDB es la elección dentro de la categoría de bases de datos no relacionales que posteriormente vamos a analizar y ver cómo se comporta con el caso de uso elegido para así llegar a la conclusión de si es la que ofrece mejores tiempos, frente a PostgreSQL y una base de datos que usa tecnología Blockchain que explicaremos más adelante.

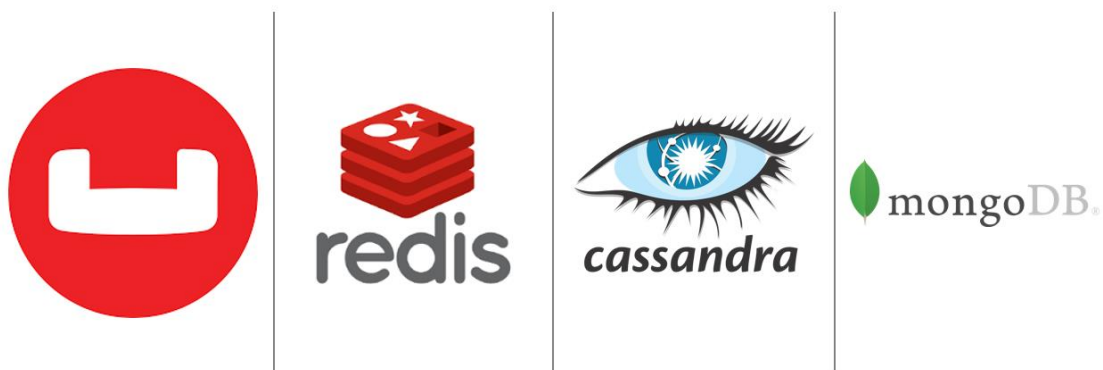


Figura 2.3: Gestores de bases de datos no relacionales (NoSQL) más usados en la actualidad.

2.3 Blockchain

Ante la presencia de bases de datos relacionales y las no relacionales, surge un nuevo tipo de almacenamiento que utiliza la tecnología Blockchain. Esta es especialmente útil cuando se requiere asegurar la persistencia y la inmutabilidad de los datos.

La tecnología Blockchain se describió por primera vez en 1991 por dos investigadores (Stuart Haber y W. Scott Stornetta) que tenían la intención de crear un sistema donde las marcas de tiempo del documento no pudieran ser modificadas. Pero no fue hasta principios de 2009, con el lanzamiento de Bitcoin, cuando Blockchain tuvo su primera aplicación en el mundo real. [7]

Este tipo de tecnología utiliza cadenas de bloques de datos, donde cada bloque se va cifrando y enlazando entre sí para proteger la seguridad y privacidad de las transacciones. Sin embargo, Blockchain tiene un requisito importante: debe haber varios usuarios (nodos) que se encarguen de verificar esas transacciones para validarlas y que así el bloque correspondiente a esa transacción quede registrado. Esto permite eliminar a los intermediarios, descentralizando toda la gestión.

El objetivo del Blockchain es permitir que la información digital se pueda distribuir y registrar, incapacitando la posibilidad de edición.

Cuando hablamos de cadenas y bloques, nos estamos refiriendo a la información digital (bloque) almacenada en una base de datos pública (cadena). Estos bloques almacenan información sobre las transacciones (fecha, hora, costo), información sobre las personas que participan en la transacción (usando firma digital, por ejemplo) e información que sirve para diferenciarse del resto de bloques (mediante el uso de códigos hash almacenados dentro de cada bloque).

Como curiosidad, un solo bloque de la cadena puede almacenar hasta 1MB de datos. Dependiendo del tamaño que tenga estas transacciones, un solo bloque podrá albergar unos pocos miles de transacciones únicamente dentro de él.

Las cadenas de bloques funcionan de la siguiente manera, cuando un bloque almacena datos nuevos, este se agrega a la cadena de bloques. Cuando un nuevo bloque es agregado a la cadena, esta se encuentra disponible para cualquier usuario que desee verlo.

Sin embargo, para que se puede agregar a esta cadena se deben de dar cuatro sucesos:

1. Se debe de dar que haya una transacción.
2. La transacción debe verificarse.
3. La transacción debe guardarse en un bloque.
4. El bloque debe disponer de un código hash único.

Además de llevar ese código hash, el bloque también contendrá el hash del bloque más reciente de la cadena. Este hash, es el resultado de haber aplicado a la información una serie de operaciones criptográficas, generando identificadores únicos e irrepetibles [8]

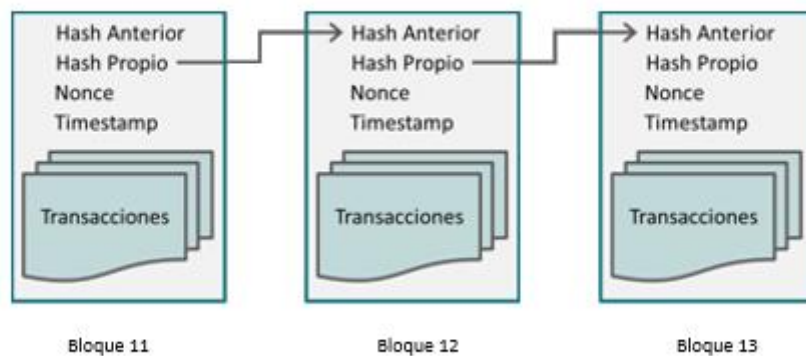


Figura 2.4: Imagen que muestra la estructura interna de cada bloque y cómo se organizan entre ellos formando la cadena de bloques.

Con respecto a la seguridad, esta tecnología explica como hace frente a los problemas de seguridad y confianza de varias formas. Primero de todo, los nuevos bloques siempre se almacenan de forma lineal y siguiendo una cronología, es decir, se agregan al final de la cadena.

Al agregar el bloque al final de esa cadena, es bastante complicado volver a algún bloque y alterar su contenido. Esto es debido al hash propio de cada bloque y al hash del bloque anterior. Si por algún casual se modificase el contenido del bloque, el hash se modificaría, incapacitándolo de la cadena.

Si se deseara cambiar un bloque, la persona que quisiera realizarlo debería cambiar todos los datos de los bloques de la cadena, para así volver a recalcular los códigos hash, sin embargo, esto requiere de muchísima potencia informática por lo que es improbable de que suceda. En definitiva, al indexar un bloque a la cadena, se vuelve difícil de modificar e imposible de eliminar.

Con respecto a la confianza, las redes Blockchain han implementado una serie de pruebas para todos aquellos que quieran unirse para agregar bloques a la cadena. Son las conocidas como modelos de concesos, donde básicamente el usuario tiene que probarse a sí mismo antes de poder participar en la red.

Esta prueba consiste en resolver un problema matemático computacional complejo que, si el ordenador resuelve algún problema, se convierte en un usuario capaz de poder agregar bloques a la cadena.

Por otro lado, tenemos Bitcoin, un término bastante relacionado con Blockchain ya que se basa en este, definido como un sistema efectivo electrónico completamente de igual a igual, sin necesidad de un tercero confiable.

Las transacciones en Bitcoin se verifican por una red de ordenadores. Por ejemplo, en el caso de que una persona desee pagar a otra usando Bitcoin, los ordenadores que se encuentran en la red compiten para verificar la transacción mediante la ejecución de un programa con el fin de resolver el problema matemático complejo comentado anteriormente. Al resolver el problema, también se habrá verificado el bloque mediante su trabajo logarítmico.

Una vez se completa la transacción, esta se registra de manera pública, se almacena en el bloque y se agrega a la cadena, haciéndola inalterable. Al acabar de verificar con éxito los bloques, son recompensados por su trabajo con criptomonedas, un tipo de moneda digital.

Sin embargo, los usuarios deben de ejecutar un programa que actúa a modo de cartera digital (wallet) para la realización de transacciones. Esta wallet está formada por dos claves criptográficas, una pública (firma digital del usuario) y otra privada.

En las transacciones, donde se ingrese o se retire lo determina esta clave pública ya que actúa a modo de ubicación. Esta clave pública es una versión reducida de la clave privada, que fue creada a través de un algoritmo matemático. Por esto se dice que Blockchain es una tecnología confidencial, ya la complejidad de estos algoritmos imposibilita revertir el proceso para la generación de la clave privada a partir de la pública.

Hoy en día cada vez más empresas están empezando a apostar por la integración de Blockchain en sus operaciones comerciales.

Una encuesta realizada por Deloitte, una red de servicios profesionales, sobre la integración de Blockchain en sus operaciones comerciales, reafirmó este hecho, de entre las 1000 compañías encuestadas, el 34% ya contaba con un sistema Blockchain en producción, mientras que el otro 41% esperaba implementarlo en los próximos 12 meses. [9]

Algunas de las aplicaciones más populares de Blockchain son: el uso bancario, uso en criptomonedas, uso sanitario, uso en registros de propiedad, uso en contratos inteligentes (este tema lo abarcaremos más adelante) y usos en la votación, entre otros.

Con respecto a las ventajas de Blockchain, esta cuenta con una precisión mejorada al eliminar la participación humana en los procesos de verificación y con una reducción de los costos mediante la eliminación de terceros. Además, la descentralización hace que sea más difícil de manipular y sus transacciones son seguras, eficientes y privadas.

Por otro lado, pese que tiene bastantes ventajas significativas, esta tecnología cuenta con obstáculos que no son sólo técnicos. Significa un gran costo tecnológico asociado con la minería de bitcoin, posee bajas transacciones por segundo, es susceptible a la piratería y posee un historial de uso relacionado a actividades ilegales, como el caso de Silk Road (mercado negro online) [10].

De entre todas las tecnologías Blockchain más usadas como Ethereum o Hyperledger, nosotros nos hemos decantado por Kaleido.

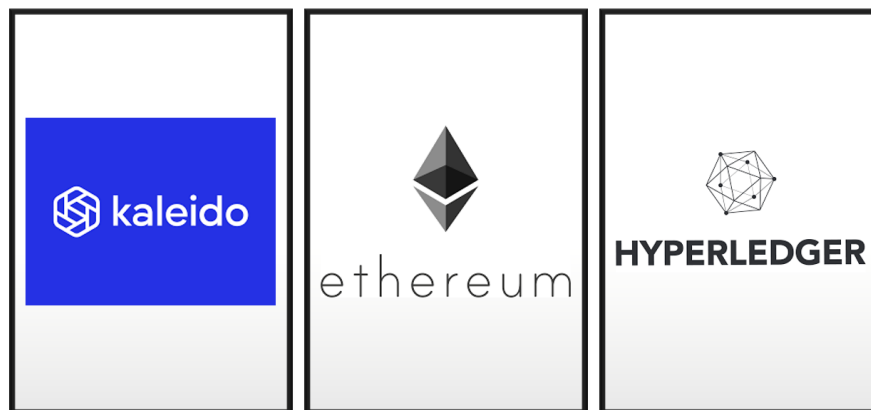


Figura 2.5: Imagen que refleja tres tecnologías Blockchain conocidas y usadas en la actualidad.

3

Análisis y organización

En este apartado comenzamos a adentrarnos en lo que es la organización del proyecto, donde hablaremos sobre los objetivos de nuestro proyecto y la metodología que hemos seguido, además de analizar los requisitos funcionales, los no funcionales y los casos de usos, junto a sus respectivos diagramas.

Las 3 páginas web que vamos a crear en este TFG con los distintos gestores de almacenamiento giran en torno a una situación en común, de los que sacamos distintos casos de uso. Con caso de uso, nos referimos a cada flujo diferente que podemos seguir dentro de la situación que analizaremos.

La situación elegida que vamos a desarrollar se trata del ciclo de vida de los animales que se encuentran en una granja, similar al sistema implementado en Carrefour con los pollos, donde se pretende dejar constancia de todo el proceso que vayan recibiendo los distintos animales que se encuentren en la granja.

Analizaremos la trayectoria del animal de la granja desde que llega hasta que es sacrificado en el matadero, almacenando cada paso que influya en el mismo. Se almacenará información acerca de la alimentación del animal, de su transporte, de las inspecciones que este reciba y de su sacrificio. También tendremos almacenado a los distintos usuarios que intervienen en el proceso, ya que contamos con un sistema de roles, que dependiendo del que tengas asignado, podrás realizar ciertas operaciones, es decir, si eres granjero, no vas a dejar constancia sobre transportes, ya que tu rol no es transportista.

También almacenaremos información sobre los tipos de animales y los animales, denominados como tipo de objeto y objeto, e información sobre las 4 operaciones habladas anteriormente, además de las distintas características que puedan tener los distintos animales de cada especie con sus valores respectivos.

3.1 Objetivos

El objetivo que pretendemos conseguir con este TFG es la de conocer cómo funcionan las 3 tecnologías que hemos escogido, en especial la tecnología Blockchain para, de este modo, familiarizarnos con su uso y evaluar su rendimiento frente a las tradicionales bases de datos relacionales y las no relacionales comúnmente conocidas como NoSQL.

Por tanto, nuestro objetivo primordial de este TFG es la de hallar cuál de las 3 opciones elegidas en este proyecto es la que mejor rendimiento proporciona para almacenar los datos.

La evaluación de los rendimientos la llevaremos a cabo mediante la implementación de una especie de banco de pruebas, donde se irán midiendo los distintos tiempos de simulación que el usuario realice tanto para inserciones como para consulta de datos en la base de datos, así como la ejecución de ambas.

3.2 Metodología usada

La metodología que hemos seguido para la elaboración del proyecto ha sido similar a un modelo de metodología en cascada, donde disponíamos de una serie de fases que se iban realizando de forma ordenada y que, posteriormente, se volverían a seguir para el resto de las implementaciones de cada base de datos.

En un primer lugar, decidimos escoger el desarrollo de la aplicación web basada en una base de datos relacional ya que disponíamos de más conocimiento sobre dichos modelos al ser los que habíamos estudiado en la carrera.

Una vez finalizada esta implementación, la siguiente elección fue adentrarnos en el aprendizaje de las dos nuevas tecnologías restantes (las bases de datos NoSQL y Blockchain, respectivamente), repitiendo las fases de la metodología en cascada.

Finalmente, realizamos una última fase de análisis, pero comparando los distintos tipos de bases de datos tratados, tal y como hemos comentado anteriormente.

3.3 Requisitos funcionales

Requisito	Título	Descripción
RF1	Creación de usuarios	La aplicación dispondrá de un apartado de registro con el que añadir información sobre nuevos usuarios.
RF2	Creación de animales	La aplicación contará con un apartado para insertar información sobre nuevos animales incluyendo características, tipo y nombre.
RF3	Creación de sacrificios	La aplicación dispondrá de un apartado con el que crear nuevas operaciones de sacrificio.
RF4	Creación de alimentaciones	La aplicación dispondrá de un apartado con el que crear nuevas operaciones de alimentaciones.
RF5	Creación de inspecciones	La aplicación dispondrá de un apartado con el que crear nuevas operaciones de inspecciones.
RF6	Creación de transportes	La aplicación dispondrá de un apartado con el que crear nuevas operaciones de transportes.
RF7	Sistema de simulación	La aplicación tendrá un apartado de simulación que generará consultas e inserciones de manera aleatoria que al finalizar muestra el tiempo que ha tardado.
RF8	Información de animales	Se mostrará información sobre las características de los animales sobre los cuáles se ha realizado alguna operación.
RF9	Consultar sacrificios	La aplicación contará con un apartado en el que poder ver la información relativa a los distintas operaciones de sacrificio realizadas por el usuario logeado.
RF10	Consultar inspecciones	La aplicación contará con un apartado en el que poder ver la información relativa a los distintas operaciones de inspección realizadas por el usuario logeado.
RF11	Consultar alimentaciones	La aplicación contará con un apartado en el que poder ver la información relativa a los distintas operaciones de alimentación realizadas por el usuario logeado.

RF12	Consultar transportes	La aplicación contará con un apartado en el que poder ver la información relativa a los distintas operaciones de alimentación realizadas por el usuario logeado.
RF13	Autenticación de usuarios	La aplicación dispondrá de un sistema de autenticación de usuarios para acceder al contenido de la web.
RF14	Sistema de roles	La aplicación estará basado en un sistema de roles asociados a los usuarios que afectará a las operaciones que puedas realizar dentro de la web. Además de esto, la aplicación contará con un rol administrador encargado de controlar las distintas consultas e inserciones que se lleven a cabo, así como el manejo de la simulación y la inserción de animales en la base de datos.

3.4 Requisitos no funcionales

Requisito	Título	Descripción
RNF1	Control de acceso	La aplicación está diseñada para que no se pueda acceder a cualquier vista si no has iniciado sesión previamente o si no dispones del rol adecuado.
RNF2	Almacenamiento	El sistema será capaz de almacenar toda la información relativa a los distintos registros, así como las inserciones resultantes de haber realizado el proceso de simulación.
RNF3	Tiempos de espera bajos	El sistema proporcionará tiempos de espera reducidos a la hora de navegar por la web e interactuar con ella.
RNF4	Implementación con cada base de datos	La aplicación está diseñada para las tres bases de datos que hemos escogido, conservando siempre las mismas funcionalidades.
RNF5	Control de duplicados	El sistema cuenta con un sistema de identificadores para poder distinguir las distintas operaciones y, en el caso de los usuarios, el identificador se trata de su DNI.

3.5 Casos de uso

Con respecto a los casos de uso, contamos con distintos casos en función de la operación que se vaya a realizar. Algunas operaciones sólo podrán llevarse a cabo con un rol específico de usuario, siendo el rol Administrador el que más operaciones puede realizar dentro de la web.

Caso de uso	CU1 - Crear Animal
Actor	Administrador
Descripción	Inserción de un nuevo animal en la base de datos, a partir de los tipos disponibles.
Flujo básico	1. Inicio de sesión Debe de haberse realizado el inicio de sesión para acceder a la funcionalidad. 2. Pestaña Animales Debe de acceder a la pestaña de Animales para poder realizar dicha operación. 3. Rellenar campos y aceptar operación Debe de rellenarse todos los campos de manera obligatoria, seleccionando tipo y cumplimentando la información relativa a su nombre y características. Tras esto, pinchar en el botón para que se lleve a cabo la operación.
Pre-condición	El rol del usuario debe ser Administrador.
Post-condición	Se inserta la información relativa al animal en la base de datos.

Caso de uso	CU2 - Registrar Usuario
Actor	Usuario no registrado
Descripción	Inserción de un nuevo usuario en la base de datos.
Flujo básico	1. Vista Registro A la hora de iniciar sesión, se debe pinchar en el enlace que te lleva a la pantalla de registro. 2. Rellenar campos y aceptar operación Debe de rellenarse todos los campos de manera obligatoria, seleccionando nombre, apellidos, dni, email, contraseña y rol. Tras esto, pinchar en el botón para que se lleve a cabo la operación.
Pre-condición	Ser un usuario no registrado (El sistema impide que se registre un usuario con un DNI que ya se encuentre en la base de datos)
Post-condición	Se inserta la información relativa al nuevo usuario en la base de datos y te redirige a la vista de inicio.

Caso de uso	CU3 - Crear Transporte
Actor	Transportista, Administrador
Descripción	Inserción de una nueva operación de transporte en la base de datos.
Flujo básico	<p>1. Inicio de sesión Debe de haberse realizado el inicio de sesión para acceder a la funcionalidad.</p> <p>2. Pestaña Insertar Debe de acceder a la pestaña de Insertar para poder crear un nuevo transporte.</p> <p>3. Rellenar campos y aceptar operación Debe de rellenarse todos los campos de manera obligatoria, seleccionando los animales transportados y cumplimentado la información relativa a la operación. Tras esto, pinchar en el botón para que se lleve a cabo la operación.</p>
Pre-condición	El rol del usuario debe ser Transportista o Administrador.
Post-condición	Se inserta la información relativa al nuevo transporte en la base de datos y te redirige a la vista de inicio.

Caso de uso	CU4 - Crear Sacrificio
Actor	Matarife, Administrador
Descripción	Inserción de una nueva operación de sacrificio en la base de datos.
Flujo básico	Mismo flujo que el CU3, pero cumplimentado la información relativa a la operación de sacrificio.
Pre-condición	El rol del usuario debe ser Matarife o Administrador. El animal no puede haber sido sacrificado previamente.
Post-condición	Se inserta la información relativa al nuevo sacrificio en la base de datos y te redirige a la vista de inicio.

Caso de uso	CU5 - Crear Inspección
Actor	Inspector, Administrador
Descripción	Inserción de una nueva operación de inspección en la base de datos.
Flujo básico	Mismo flujo que el CU3, pero cumplimentado la información relativa a la operación de inspección.
Pre-condición	El rol del usuario debe ser Inspector o Administrador.
Post-condición	Se inserta la información relativa a la nueva inspección en la base de datos y te redirige a la vista de inicio.

Caso de uso	CU6 - Crear Alimentación
Actor	Granjero, Administrador
Descripción	Inserción de una nueva operación de alimentación en la base de datos.
Flujo básico	Mismo flujo que el CU3, pero cumplimentado la información relativa a la operación de alimentación.
Pre-condición	El rol del usuario debe ser Granjero o Administrador.
Post-condición	Se inserta la información relativa a la nueva alimentación en la base de datos y te redirige a la vista de inicio.

Caso de uso	CU7 - Consultar Transporte
Actor	Transportista, Administrador
Descripción	Consultar información sobre los distintos transportes realizados por tu usuario. En el caso del administrador, este podrá ver la información de todos los transportes de la base de datos.
Flujo básico	1. Inicio de sesión Debe de haberse realizado el inicio de sesión para acceder a la funcionalidad. 2. Pestaña Consultar Debe de acceder a la pestaña de Consultar para poder ver la información de los transportes.
Flujo alternativo	1. Consultar Animales Transportados En la pestaña Consultar, se puede ver la información de los distintos animales transportados al pinchar en el identificador del transporte. 2. Consultar Características En la vista de los animales transportados, se puede ver las características de un animal al pinchar en su identificador.
Pre-condición	El rol del usuario debe ser Transportista o Administrador.
Post-condición	

Caso de uso	CU8 - Consultar Sacrificio
Actor	Matarife, Administrador
Descripción	Consultar información sobre los distintos sacrificios realizados por tu usuario. En el caso del administrador, este podrá ver la información de todos los sacrificios de la base de datos.
Flujo básico	Mismo flujo que el CU7, pero viendo la información de los sacrificios.
Flujo alternativo	1. Consultar Características En esa misma vista, si se selecciona el identificador del animal, se mostrará en otra página sus características.
Pre-condición	El rol del usuario debe ser Matarife o Administrador.
Post-condición	

Caso de uso	CU9 - Consultar Inspección
Actor	Inspector, Administrador
Descripción	Consultar información sobre las distintas inspecciones realizadas por tu usuario. En el caso del administrador, este podrá ver la información de todas las inspecciones de la base de datos.
Flujo básico	Mismo flujo que el CU7, pero viendo la información de las inspecciones.
Flujo alternativo	Mismo flujo que el CU8.
Pre-condición	El rol del usuario debe ser Inspector o Administrador.
Post-condición	

Caso de uso	CU10 - Consultar Alimentación
Actor	Granjero, Administrador
Descripción	Consultar información sobre las distintas alimentaciones realizadas por tu usuario. En el caso del administrador, este podrá ver la información de todas las alimentaciones de la base de datos.
Flujo básico	Mismo flujo que el CU7, pero viendo la información de las inspecciones.
Flujo alternativo	Mismo flujo que el CU8.
Pre-condición	El rol del usuario debe ser Granjero o Administrador.
Post-condición	

Caso de uso	CU11 - Generar Simulación
Actor	Administrador
Descripción	Realizar simulación a partir del número de inserciones y consultas deseado.
Flujo básico	<p>1. Inicio de sesión Debe de haberse realizado el inicio de sesión para acceder a la funcionalidad.</p> <p>2. Pestaña Simular Debe de acceder a la pestaña de Simular para poder llevar a cabo la simulación</p> <p>3. Rellenar campos y aceptar operación Debe de rellenarse todos los campos de manera obligatoria, seleccionando el número de inserciones y consultas que se quiera generar (cero en el caso de no querer realizar alguna). Tras esto, pinchar en el botón para que se genera la simulación.</p>
Pre-condición	El rol del usuario debe ser Administrador.
Post-condición	Se insertan registros aleatorios en la base de datos entre las distintas posibilidades (crear usuarios, crear operaciones, crear animal), se realizan las consultas y se actualiza el valor del tiempo de ejecución que quedará reflejado en la página.

3.6 Diagrama de caso de uso

Con respecto a los diagramas de caso de uso, hemos englobado los casos anteriores y los hemos agrupado según el tipo de usuario, siendo estas el rol Administrador, el rol Transportista y otra tercera opción que englobaría a los roles Inspector, Granjero y Matarife ya que sus diagramas son similares. Para estos casos de uso, el usuario deberá haber iniciado sesión previamente, por lo que omitiremos este paso.

Por ello, vamos a comenzar con el diagrama de uso para un usuario que acceda por primera vez a la web y no tenga cuenta con la que acceder, por lo que deberá realizar el registro.

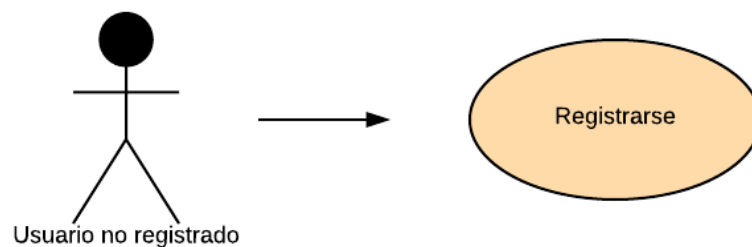


Figura 3.1: Diagrama de caso de uso para un usuario que no se encuentra registrado en la base de datos.

Tras esto, los siguientes casos de uso parten de un inicio de sesión previo por lo que no quedará reflejado en el diagrama. Cabe destacar que el administrador engloba los cuatro roles que se encuentran en la base de datos.

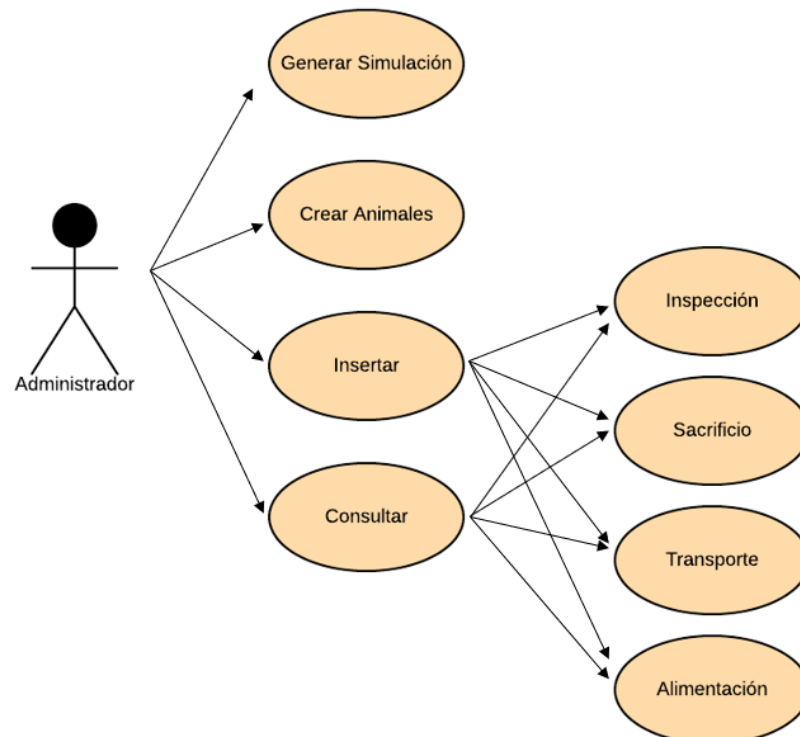


Figura 3.2: Diagrama de caso de uso para un usuario con rol Administrador.

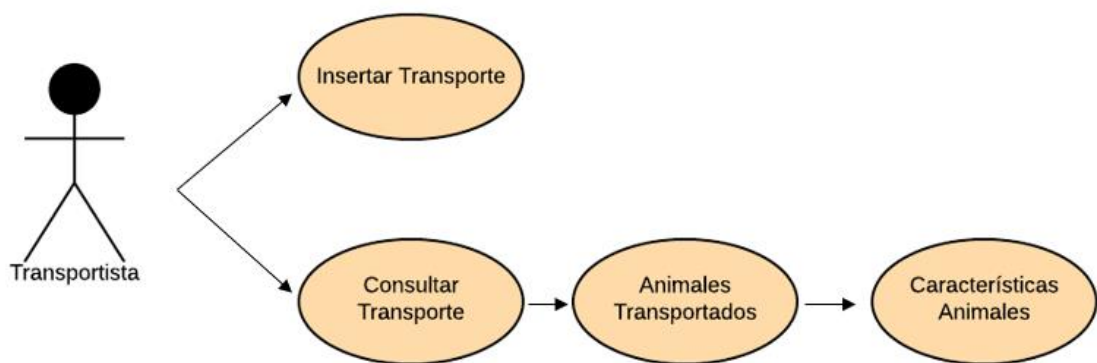


Figura 3.3: Diagrama de caso de uso para un usuario con rol Transportista

A continuación, en la Figura 3.4 se muestra el diagrama de caso de uso para un usuario con rol Granjero, Matarife o Inspector, que inserta o consulta sus respectivas operaciones, ya que tienen el mismo diagrama. La única diferencia con el rol Transportista es que en la consulta se muestre al animal afectado, en vez de tener que pinchar en el identificador de la operación para acceder a dicha información.

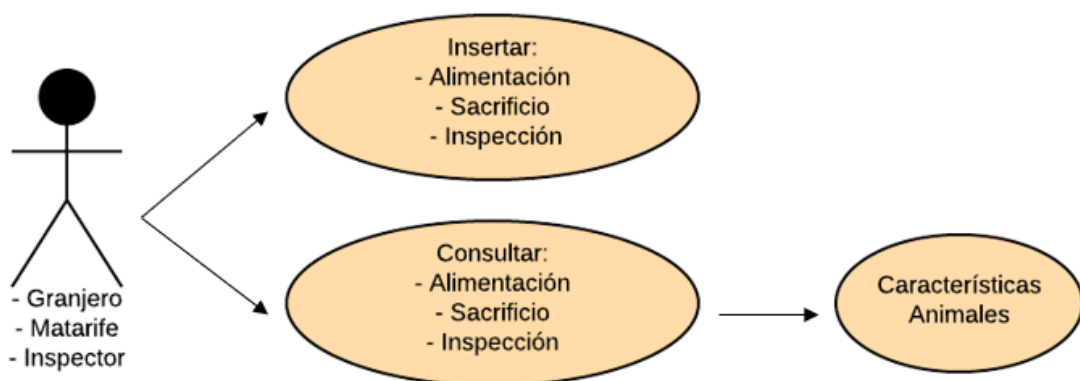


Figura 3.4: Diagrama de caso de uso para un usuario con rol Granjero, Matarife o Inspector.

4

Diseño e implementación

En este apartado, veremos las tres tecnologías que hemos seleccionado para comparar. Estas son PostgreSQL por la parte de bases de datos relacionales, MongoDB en lo relacionado a bases de datos no relacionales y en lo referente a tecnología Blockchain, nos encontramos con Kaleido.

El contenido de este apartado estará estructurado de la siguiente manera:

- Descripción de la herramienta escogida.
- Diseño de cómo está estructurada la base de datos (modelo usado).
- Explicación sobre la implementación de la misma.

4.1 PostgreSQL

4.1.1 Descripción

PostgreSQL es un sistema gestor de bases de datos relacionales y orientado a objetos reconocido como uno de los más potentes del mercado. También es multiplataforma (funciona en Windows, Linux, MacOS y otros sistemas operativos), extensible (podemos añadir funcionalidades que no vengan de serie), es escalable y funciona bajo licencia libre, es decir, se puede usar para cualquier propósito, contando con una gran comunidad que se encargan de su mantenimiento y respaldo (PGDG - PostgreSQL Global Development Group).

PostgreSQL surgió a mediados de los 80 a partir de Ingres (otro proyecto de bases de datos de la década anterior) encabezado por Michael Stonebraker, quien decidió darle el nombre de Postgres (Post Ingres). En 1989, salió la primera versión de cara al público de acceso limitado. El equipo se separó en 1994 pero pocos años después, el proyecto fue retomado y relanzado con soporte SQL, y fue en 1997 cuando ya recibió el nombre que conocemos hoy en día. [11]

PostgreSQL cuenta con una serie de características que la hacen destacar, de ahí que sea una de las más usadas en la actualidad. En primer lugar, presenta un sistema de alta concurrencia, denominado MVCC, que permite que mientras un proceso está escribiendo de una tabla, otros puedan acceder a esa tabla sin necesidad de bloqueo, dotando a cada uno de una visión consistente.

En segundo lugar, nos encontramos con el sistema “Host Standby”, que permite al usuario poder conectarse al servidor y ejecutar búsquedas en la base de datos mientras esta se encuentra en modo recuperación. También cuenta con soporte nativo para tipos de datos como textos de largo ilimitado, números de precisión arbitraria, direcciones MAC, arrays, tipos de datos propios de los usuarios o protocolos de direccionamiento IP.

Posee la capacidad de registrar cada transacción en un WAL (Write Ahead Log o Registro de Escritura Anticipada), que es un método estándar para garantizar la integridad de los datos. Este registro permite restaurar la base de datos a cualquier punto previamente guardado, haciendo posible que no sea necesario estar haciendo respaldos completos frecuentemente.

La última característica que resaltaremos es el uso de disparadores (triggers), que son procesos almacenados que se ejecutan, basados en una acción determinada sobre una tabla específica de la base de datos. Estos se definen en función de su nombre, el momento del arranque, el evento disparador, la tabla donde se ejecuta, la frecuencia de ejecución y la función llamada.

Es usado en diversos ámbitos como por ejemplo para el almacenamiento de datos (data warehousing), en servicios como Amazon Web Services Redshift (para procesamiento de datos almacenados tanto en la propia instancia como en otros servicios que puedan conectarse), en sistemas de información geográfica (servicio de mapas web), en bases de datos para servicios web como Wordpress y en una plataforma como servicio (Skype, Telefónica y BBVA, son algunos ejemplos de empresas que usan PostgreSQL como PaaS) [12].

4.1.2 Modelo

El modelo de esta base de datos está formado por 9 entidades o tablas principales, reflejadas en el modelo entidad relación de la figura 4.1.

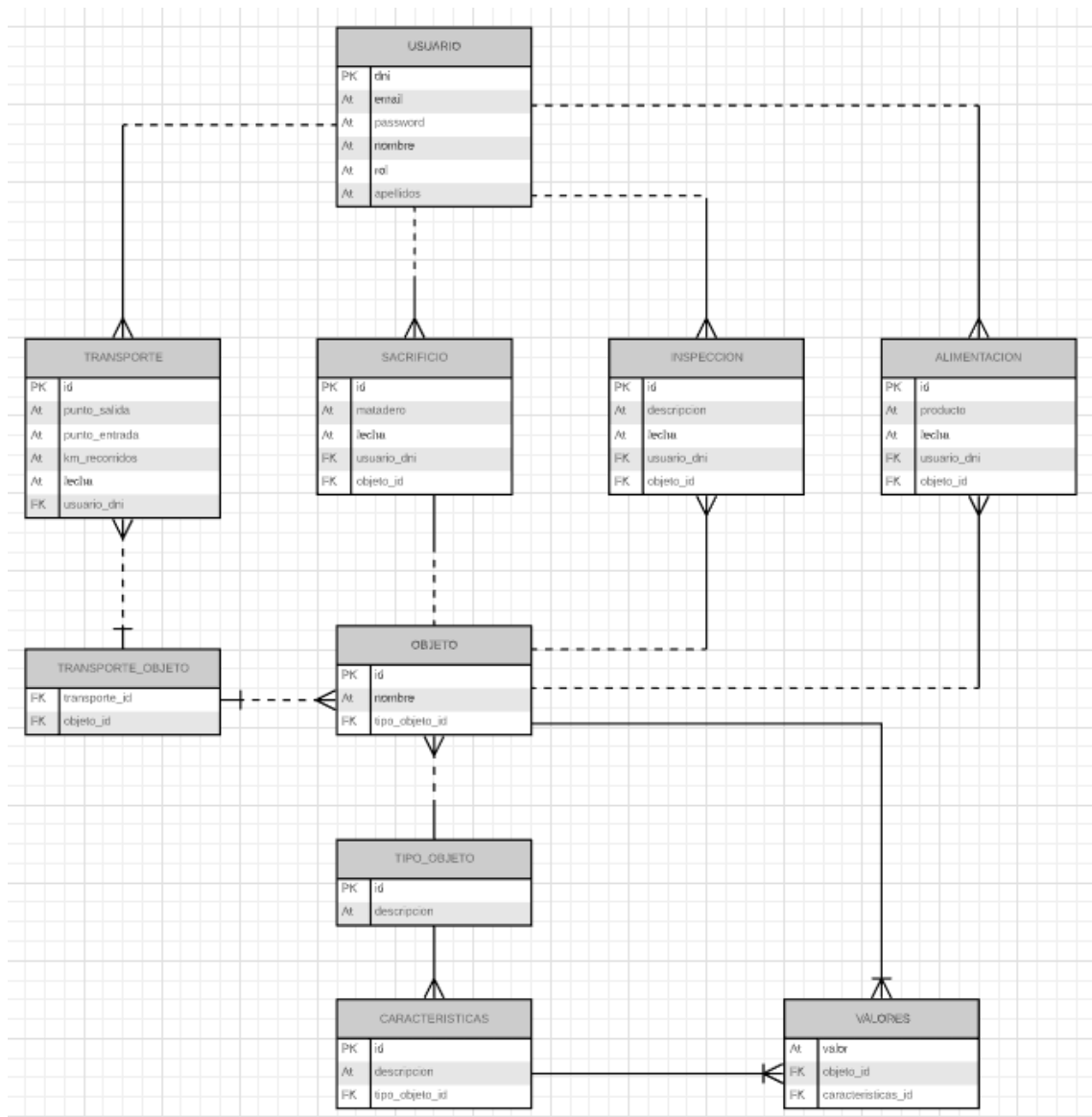


Figura 4.1: Imagen que refleja el modelo entidad relación de nuestra base de datos (PK: Primary Key o Clave primaria, At: Atributo, FK: Foreign Key o Clave foránea) y las relaciones entre las tablas.

Cabe decir que existe una tabla extra, surgida de la relación de N:N entre Transporte y Objeto, que contiene claves foráneas que hacen referencia a las claves principales de cada tabla (transporte_id y objeto_id).

4.1.3 Implementación

Las herramientas usadas para implementar la web han sido pgAdmin 4 (interfaz gráfica de PostgreSQL donde hemos configurado la base de datos y hemos creado las distintas tablas), NetBeans IDE (herramienta que hemos usado para crear la página web) y GitHub (repositorio usado para llevar un control de versiones).

Al instalarnos PostgreSQL, este nos traía una interfaz llamada pgAdmin que usamos para la gestión de la base de datos. Tras crear la conexión con el servidor (PostgreSQL 10) y crear nuestra base de datos (TFGbd), creamos las tablas y las relaciones entre estas.

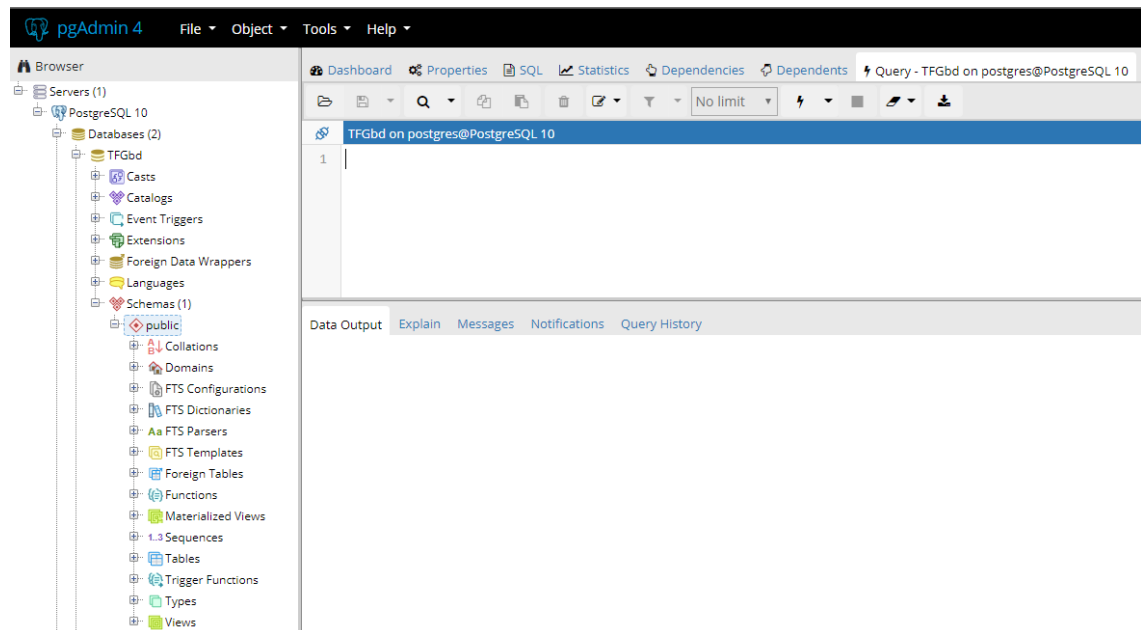


Figura 4.2: Interfaz del pgAdmin 4 que muestra como está organizada la base de datos.

Dentro de esta interfaz podemos ver información relevante sobre las distintas tablas y sus parámetros, las restricciones creadas, índices, etc. Para poder ejecutar las sentencias SQL, este cuenta con una pestaña Query Tool que se encuentra al hacer click derecho sobre el esquema público.

Con respecto a NetBeans, la distribución del proyecto se estructura en dos partes importantes: front-end y back-end. Cabe decir que hemos usado GlassFish como servidor de la aplicación.

Hemos usado el famoso patrón de arquitectura software denominado Modelo-Vista-Controlador (MVC), utilizando componentes para separar la lógica de la aplicación de la lógica de la vista. Con respecto a las vistas, hemos usado componentes de Java Server Faces (JSF), junto a componentes del framework Primefaces que hemos tenido que importar. Las vistas se tratan de archivos XHTML que cuentan con JavaScript y CSS.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xml:lang="es" lang="es"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
```

Figura 4.3: Estructura del proyecto: Anotaciones usadas para identificar los componentes JSF y los componentes de Primefaces.

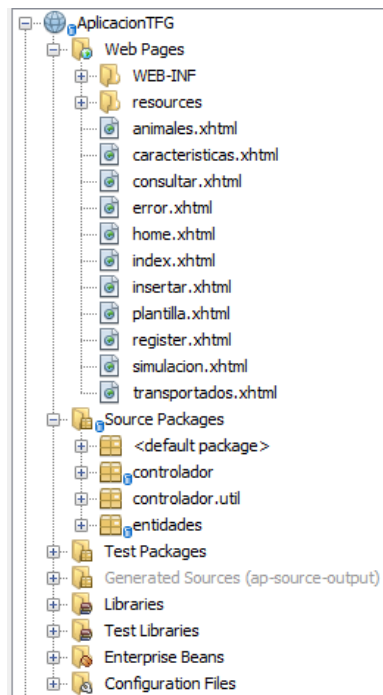


Figura 4.4: Estructura del proyecto: Organización del proyecto PostgreSQL dentro de NetBeans.

Con respecto a los modelos y controladores, ambos están escritos en lenguaje Java. Los modelos han sido mapeados mediante la conexión del proyecto con la base de datos de PostgreSQL. Esta conexión consiste en crear una unidad de persistencia, para generar un DataSource de la base de datos con el que poder establecer la conexión mediante el jdbc de PostgreSQL (un archivo con una serie de parámetros de conexión: puerto, base de datos, usuario, contraseña, url, nombre del servidor). Por otro lado, los controladores también se han generado a partir de esta conexión distribuyéndose en dos tipos de archivos, controladores y facades. Para el caso de las entidades, vemos que se identifican mediante el uso de la anotación `@Entity`, junto a su nombre y una serie de consultas que puedes realizar.

```
@Entity
@Table(name = "usuario")
@XmlRootElement
@NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u")
, @NamedQuery(name = "Usuario.findByDni", query = "SELECT u FROM Usuario u WHERE u.dni = :dni")
, @NamedQuery(name = "Usuario.findByNombre", query = "SELECT u FROM Usuario u WHERE u.nombre = :nombre")
, @NamedQuery(name = "Usuario.findByApellidos", query = "SELECT u FROM Usuario u WHERE u.apellidos = :apellidos")
, @NamedQuery(name = "Usuario.findByEmail", query = "SELECT u FROM Usuario u WHERE u.email = :email")
, @NamedQuery(name = "Usuario.findByRol", query = "SELECT u FROM Usuario u WHERE u.rol = :rol")
, @NamedQuery(name = "Usuario.findByPassword", query = "SELECT u FROM Usuario u WHERE u.password = :password"))
```

Figura 4.5: Estructura del proyecto: Anotaciones usadas para identificar una entidad, en este caso, la de Usuario.

Para los controladores, mediante la anotación `@EJB` identificamos al facade y nos comunicamos con el EntityManager (interfaz usada para interactuar con el contexto de persistencia) y en los controladores, usamos las anotaciones que hacen referencia al tipo de Java Bean y a su nombre.

En el caso de la Figura 4.6, se trata de un bean de sesión. Este tipo permite que los datos se mantengan almacenados en la clase durante toda la sesión de la aplicación.

```
@Named("usuarioController")
@SessionScoped
public class UsuarioController implements Serializable {

    @EJB
    private controlador.UsuarioFacade ejbFacade;
```

Figura 4.6: Estructura del proyecto: Anotaciones usadas para identificar un controlador, en este caso, el de Usuario.

La clase facade que se comunica con el controlador cuenta con una anotación que especifica su estado y con una anotación identifica al contexto de persistencia.

```
@Stateless
public class UsuarioFacade extends AbstractFacade<Usuario> {

    @PersistenceContext(unitName = "AplicacionTFGPU")
    private EntityManager em;
```

Figura 4.7: Estructura del proyecto: Anotaciones usadas para identificar el estado y el contexto de persistencia de los archivos facade, en este caso, el de Usuario.

4.2 MongoDB

4.2.1 Descripción

MongoDB es una base de datos distribuida, de código abierto y basada en documentos, lo que significa que los datos se almacenan en forma de documentos. En lugar de guardar los datos en las tablas como en el caso de PostgreSQL, estos se almacenan en estructuras de datos BSON (similar a JSON) con un esquema dinámico, permitiendo una rápida y sencilla integración de los datos.

En 2007, 10gen Inc. comenzó con su desarrollo mientras trabajaban en el desarrollo de una plataforma como servicio y en 2009 ya fue lanzado como producto independiente y bajo licencia de código abierto.

Entre sus características principales destacan la indexación de los campos de un documento, el soporte de búsquedas por campos, consultas de rangos y expresiones regulares, la posibilidad de escalar horizontalmente balanceando la carga o replicando los datos para mantener el funcionamiento del sistema ante fallos del hardware, el aprovechamiento de dicho balanceo de carga para el uso de MongoDB como sistema de archivos, la posibilidad de realizar consultas mediante Javascript, entre otros.

Su uso suele estar focalizado en entornos que lo necesitan para almacenamiento y registros de eventos, para comercio electrónico, para aplicaciones móviles y juegos, para manejo de estadísticas en tiempo real, para sistemas con alto volumen de lecturas y para proyectos que hacen uso de metodologías de desarrollo ágiles o iterativos.

4.2.2 Modelo

El modelo de la base de datos para MongoDB está estructurado en tres colecciones (similares a las tablas de las bases de datos relacionales). Contamos con las colecciones:

- **Usuario:** contiene la información referente al usuario: DNI, nombre, apellidos, email, password y rol.
- **Operación:** colección que engloba la información referente a los cuatro tipos de operaciones disponibles (transporte, sacrificio, inspección y alimentación).
- **Objeto:** contiene la información referente al animal: id, nombre, tipo, características y valores.

La principal diferencia con el modelo relacional es que hemos englobado las distintas operaciones en una sola colección, la cual contendrá todos los campos referentes a cada una y los documentos se diferenciarán entre sí mediante un atributo denominado tipo. También hemos unificado las tablas que contenían información sobre el animal, el tipo, las características del tipo y sus valores, que teníamos en el modelo relacional, en una única colección. Todo esto ha sido posible gracias a la dinamicidad que aporta el esquema de MongoDB.

4.2.3 Implementación

Con respecto a su implementación, para la base de datos hemos tenido que usar la consola de comandos que incluye MongoDB ya que no dispone de interfaz y hemos usado, al igual que con PostgreSQL, NetBeans IDE como herramienta para crear nuestra aplicación y GitHub como repositorio para llevar un control de versiones.

Para el caso de MongoDB, la instalación no contaba con una interfaz gráfica, sino que se hace uso de dos consolas de comandos. La primera, denominada mongod, sirve para iniciar el servidor. Sin la ejecución de ella, no podemos iniciar la segunda consola de comandos y por tanto no podemos realizar las operaciones con nuestra base de datos.

```
mongod
2020-02-01T19:03:59.988+0100 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2020-02-01T19:03:59.989+0100 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2020-02-01T19:03:59.989+0100 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2020-02-01T19:03:59.990+0100 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2020-02-01T19:03:59.991+0100 I CONTROL [initandlisten]
2020-02-01T19:04:00.010+0100 I SHARDING [initandlisten] Marking collection local.system.replset as collection version: <unsharded>
2020-02-01T19:04:00.063+0100 I STORAGE [initandlisten] Flow Control is enabled on this deployment.
2020-02-01T19:04:00.064+0100 I SHARDING [initandlisten] Marking collection admin.system.roles as collection version: <unsharded>
2020-02-01T19:04:00.065+0100 I SHARDING [initandlisten] Marking collection admin.system.version as collection version: <unsharded>
2020-02-01T19:04:00.084+0100 I SHARDING [initandlisten] Marking collection local.startup_log as collection version: <unsharded>
2020-02-01T19:04:00.460+0100 W FTDC [initandlisten] Failed to initialize Performance Counters for FTDC: WindowsPdhError: PdhExpandCounterPathW failed with 'El objeto especificado no se encontró en el equipo.' for counter '\Memory\Available Bytes'
2020-02-01T19:04:00.460+0100 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'D:/data/db/diagnostic.data'
2020-02-01T19:04:00.465+0100 I SHARDING [LogicalSessionCacheRefresh] Marking collection config.system.sessions as collection version: <unsharded>
2020-02-01T19:04:00.465+0100 I NETWORK [initandlisten] Listening on 127.0.0.1
2020-02-01T19:04:00.465+0100 I SHARDING [LogicalSessionCacheReap] Marking collection config.transactions as collection version: <unsharded>
2020-02-01T19:04:00.467+0100 I NETWORK [initandlisten] waiting for connections on port 27017
```

Figura 4.8: Vista de la consola de comandos del servidor de MongoDB.

La segunda consola es la consola que se utiliza para la gestión de la base de datos, llamada mongo. Es en esta consola donde creamos un nuevo database y las distintas colecciones ya que, a diferencia de PostgreSQL, aquí no se trabaja con tablas. Con respecto a la introducción de los datos, los hacemos siguiendo la estructura de los documentos BSON.

```
mongo
2020-02-01T19:03:59.988+0100 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2020-02-01T19:03:59.988+0100 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2020-02-01T19:03:59.988+0100 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2020-02-01T19:03:59.989+0100 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2020-02-01T19:03:59.989+0100 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2020-02-01T19:03:59.990+0100 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2020-02-01T19:03:59.991+0100 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> use tfg;
switched to db tfg
> show collections;
objeto
operacion
usuario
```

Figura 4.9: Vista de la consola de comandos para la gestión de la base de datos MongoDB.

La creación de las colecciones se puede llevar a cabo ya sea creándolas directamente o bien, insertando documentos con el nombre de la nueva colección, tal y como se muestra en la figura 4.10.


```
// Crear collection
db.createCollection("usuarios")

// O bien se puede insertar filas dentro de una coleccion aun no creada para que se cree
// de forma automatica
db.usuarios.insert({nombre: "Alberto", pais: "España"})
```

Figura 4.10: Creación de una colección de las dos formas posibles.

Con respecto a NetBeans, la distribución del proyecto es similar que la de PostgreSQL y seguimos usando GlassFish como servidor de la aplicación. También volvemos a hacer uso del patrón Modelo-Vista-Controlador (MVC), hemos usado componentes de JSF, junto a componentes de Primafaces y los lenguajes usados son los mismos.

Las únicas diferencias con respecto al anterior proyecto consisten en la disposición de los controladores y las entidades, ya que para este caso no hemos necesitado crear el mismo número de colecciones que de tablas en PostgreSQL, y en cómo hemos tenido que configurar las conexiones y mapeados de estas.

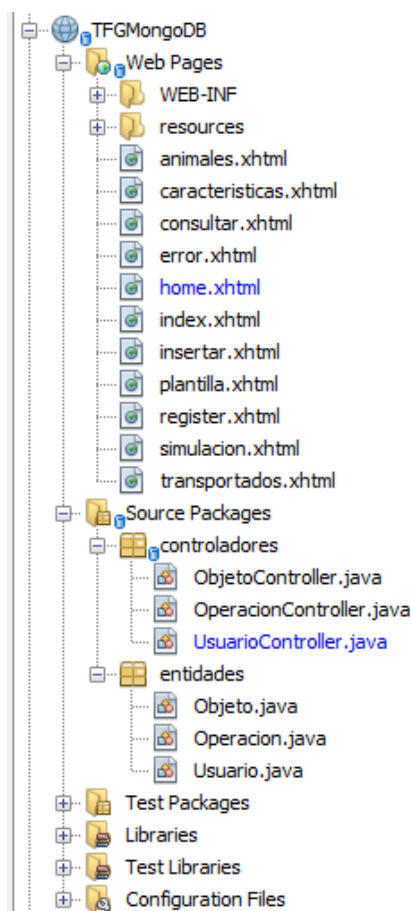


Figura 4.11: Estructura del proyecto: Organización del proyecto MongoDB dentro de NetBeans.

Para el caso de las entidades, no contamos con ningún tipo de anotación específica como en el caso de PostgreSQL. Se trata de una clase Java con su respectiva declaración de variables, constructor y funciones.

En el caso de los controladores, en este caso no contamos con ningún elemento facade, volvemos a hacer uso de los Java Beans con sus respectivas etiquetas que determinan el nombre y su tipo. La diferencia con respecto a los otros controladores, es que en estas clases se declaran variables de conexión con la base de datos con información acerca del cliente de mongo, la database usada y las distintas colecciones.

```
@Named("usuarioController")
@SessionScoped
public class UsuarioController implements Serializable {

    private MongoClient cliente = new MongoClient("localhost", 27017);
    private MongoDBDatabase dbs = cliente.getDatabase("tfg");
    private MongoCollection<Document> collection = dbs.getCollection("usuario");
    private MongoCollection<Document> collectionOperacion = dbs.getCollection("operacion");
    private MongoCollection<Document> collectionObjeto = dbs.getCollection("objeto");
```

Figura 4.12: Estructura del proyecto: Declaración de variables de conexión de MongoDB en un controlador.

Como hemos comentado anteriormente, hemos usado GitHub como repositorio donde guardar nuestros archivos y poder recuperarlos en caso de pérdidas, además de llevar un control de los distintos cambios realizados sobre la aplicación.

Decidimos usar GitHub ya que estábamos acostumbrados a manejarlo en la carrera en distintas asignaturas y porque NetBeans cuenta con la opción de conectarse con GitHub de forma muy sencilla e intuitiva, facilitándonos el proceso y ahorrándonos tener que hacer uso de la consola de comandos de GitHub. En la figura 4.13 vemos cómo estaría estructurado nuestro repositorio privado de MongoDB.

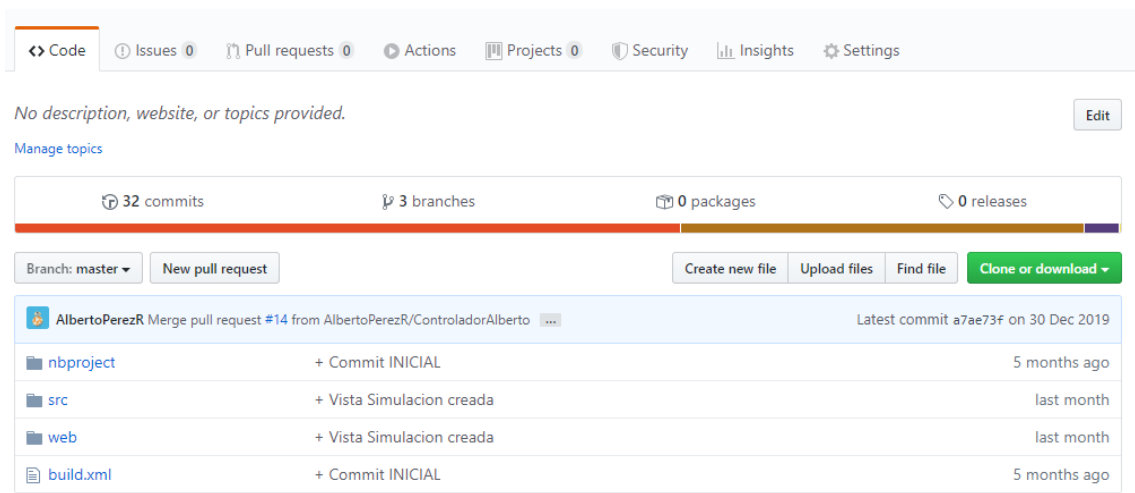


Figura 4.13: Repositorio privado de GitHub para el proyecto de MongoDB.

4.3 Kaleido

4.3.1 Descripción

Por último, nos encontramos con la gran novedad, Kaleido, que ofrece un software como servicio (SaaS) completo con el que poder crear, operar y escalar soluciones de Blockchain empresariales. Kaleido acelera todo el viaje desde una prueba de concepto (PoC) a redes en vivo con una rápida incorporación de miembros, una reforzada escala empresarial, un análisis y un gobierno compartido flexible de tecnología de la información (TI) [13].

La prueba de concepto no es más que una implementación incompleta o resumida de una idea, un método, con el fin de poder verificar que esa idea pueda llegar a ser explotada útilmente.

Con respecto a su historia, todo comenzó cuando una startup empresarial de ConsenSys, llamada Kaleido, se asocia con dos grandes tecnologías que colectivamente abarcan el 80% de la nube pública (Microsoft Azure y Amazon AWS). Con este movimiento, pretendieron dar a conocer que, dentro del Blockchain empresarial, se puede contemplar el desarrollo de consorcios y grupos de usuarios sin tener que prescindir de la infraestructura tecnológica y los estándares de nube preferidos.

Nosotros en concreto, hemos usado Kaleido junto a Ethereum por debajo como Blockchain ya que se nos permitía su integración, pero esto lo hablaremos en el apartado de implementación. [14]

Entre las características de Kaleido destacan:

- **Redes instantáneas:** Posibilidad de crear y personalizar las cadenas privadas de forma sencilla, reduciendo el costo y la complejidad de realizar su construcción desde cero.
- **Cadenas sin límites o Borderless chain:** Capacidad para implementar y ejecutar redes híbridas a través de proveedores en la nube, o bien, desplegándolas bajo tu propio firewall.
- **Tokenización de activos:** Con la generación de tokens de Kaleido, se puede digitalizar y fraccionar activos a través de esta tokenización para cualquier caso de uso.
- **Master Shared-IT:** Permite escalar la red y gobernar con facilidad construyendo un gobierno distribuido, gracias a la incorporación simplificada de miembros, de herramientas de gobierno integradas y de flujos de trabajo automatizados [15].

4.3.2 Modelo

En el caso de Blockchain, no disponemos de un modelo de base de datos ya que la información se almacena cifrada en los bloques ya sea enviando información en transacciones o enviando unos tipos especiales de estructura denominados contratos inteligentes (Smart Contracts). [16]

Hemos creado tres contratos inteligentes que siguen una idea similar a las colecciones de MongoDB:

- **Usuario:** Contrato que contiene información sobre el DNI, el nombre, los apellidos, el email, la password y el rol del usuario, así como la dirección (clave hexadecimal) del usuario que ha iniciado sesión.
- **Objeto:** Contrato que contiene la información relevante al animal (nombre, tipo) y a sus características (par atributo valor).
- **Operación:** Contrato donde se almacena información sobre los transportes (fecha, DNI, punto de entrada, punto de salida, km recorridos, animales transportados), los sacrificios (fecha, DNI, matadero, animal), las inspecciones (fecha, DNI, descripción, animal) y las alimentaciones (fecha, DNI, producto, animal).

4.3.3 Implementación

Cabe decir, que, en este caso, las herramientas usadas para llevar a cabo la implementación de la web y de configuración de Blockchain han sido totalmente distintas a los dos tipos de bases de datos anteriores. Usando para la configuración de Kaleido su propia web, para la elaboración de la web hemos usado el editor Sublime Text y hemos vuelto a usar GitHub para salvaguardar los archivos y llevar un control de versiones.

Para poder ejecutar nuestra aplicación, hemos tenido que instalarnos la consola de comandos de Node.js, ya que sólo disponíamos de un editor de texto, y en esa consola hemos tenido que instalar diversos paquetes como por ejemplo el Framework Truffle, encargado de facilitarnos la administración de los contratos.

Con respecto al desarrollo de la aplicación, hemos tenido que usar un nuevo lenguaje de programación, denominado Solidity, para la elaboración de los Smart Contracts [17]. También hemos usado Node.js para la elaboración de las funcionalidades de la aplicación, CSS para darle estilos, HTML para las vistas y hemos introducido Bootstrap para mejorar las vistas ya que no podíamos usar el framework Primefaces como anteriormente habíamos hecho debido a su incompatibilidad.

Por tanto, las vistas ahora se comunican con un archivo javascript encargado de las funcionalidades, donde aplicamos jQuery y Nodejs, en lugar de comunicarnos con controladores como en los dos anteriores casos.

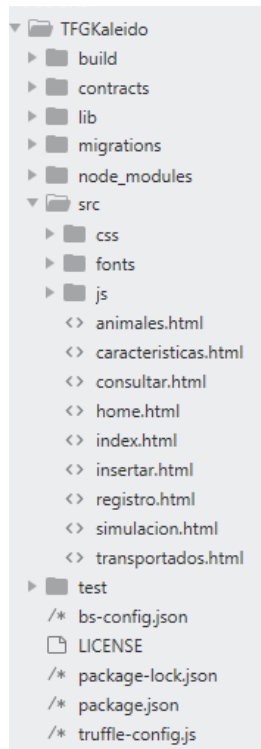


Figura 4.14: Estructura del proyecto: Organización del proyecto Kaleido dentro de Sublime Text.

Los contratos se encuentran en el directorio contracts, que necesitan de los archivos de la carpeta migrations para poder desplegarlos. Las vistas, los css y el js se encuentran dentro de la carpeta src, mientras que los archivos de configuración se encuentran al final del todo. De entre ellos, el más importante es el truffle-config.js que usamos para establecer la conexión con el nodo de Kaleido.

A continuación, vamos a ver en menor medida cómo se encuentran contruidos los distintos elementos que intervienen en el proyecto. El primer elemento son los contratos, escritos en Solidity, que es un lenguaje bastante parecido a Java, salvo por la incompatibilidad con algunos tipos de datos. Tendríamos las estructuras donde guardamos la información, nuestro constructor y las pertinentes funciones que se necesitan, por ejemplo, el contrato de Objeto dispondrá de una función para crear animales

```
function creaAnimal(string memory _nombre, string memory _tipo) public returns (uint animalID){
    // Seteamos valores
    animales[numAnimales] = Animal(_nombre, _tipo, 0);

    animalID = numAnimales;

    numAnimales++;
}
```

Figura 4.15: Estructura del proyecto: Ejemplo de funciones de los contratos (en este caso de Objeto).

El segundo elemento son los archivos de migración que debe de haber dos, el inicial que se encarga de configurar un contrato, y el segundo donde debes introducir el nombre de los distintos contratos que vayas a desplegar.

```

1 |const Usuario = artifacts.require("Usuario");
2 |const Objeto = artifacts.require("Objeto");
3 |const Operacion = artifacts.require("Operacion");
4
5 |module.exports = function(deployer) {
6 |  deployer.deploy(Usuario);
7 |  deployer.deploy(Objeto);
8 |  deployer.deploy(Operacion);
9 |};

```

Figura 4.16: Estructura del proyecto: Archivo de migración encargado de desplegar los contratos.

Y por último contaríamos con las vistas, que son similares a las anteriores, y con nuestro JavaScript encargado de comunicarse con la vista. Esta se llama nada más comenzar a ejecutarse la aplicación haciendo que se cargue el archivo, que se declara con el nombre App. Dentro de este App, contamos con funciones encargadas para la configuración de las conexiones con Kaleido (a través del usuario, contraseña y endpoint), para la carga de los contratos y una última función encargada de manejar los distintos eventos que puedan llegar a ocurrir en la página. Dentro de este archivo, haremos uso de las funciones de manera síncrona o asíncrona en función de nuestras necesidades. En la figura 4.17 vemos la forma en que los datos son introducidos en el contrato, donde se especifica además el usuario que lo envía y la cantidad de gas necesaria para enviar la transacción.

```

var account = accounts[0];
App.contracts.Objeto.deployed().then(async function(instance) {
  objetoInstance = instance;

  // Creamos animal
  await objetoInstance.creaAnimal(nombre, tipo, {from: account, gas:3000000});

```

Figura 4.17: Estructura del proyecto: Ejemplo de llamada a una función de un contrato.

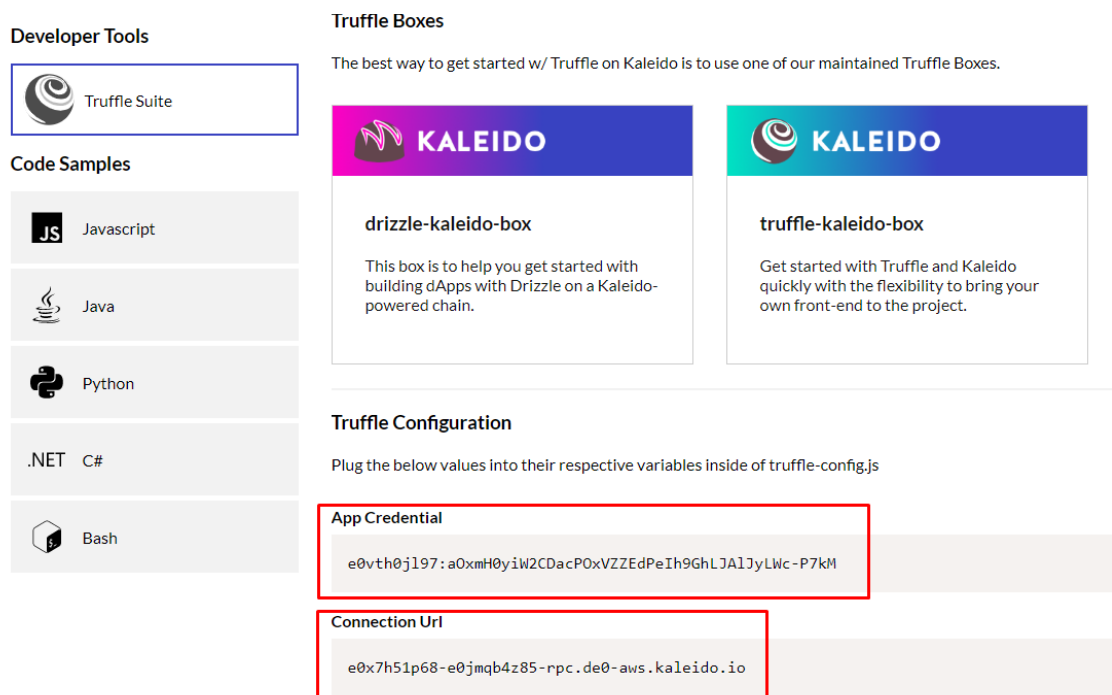
A continuación, vamos a hablar de cómo hemos configurado Kaleido. Para que todo este proceso se pudiera llevar a cabo, teníamos que disponer de una red Blockchain donde hubiera nodos participantes con sus respectivas carteras (wallet) con fondos.

The screenshot shows the Kaleido console interface. At the top, there's a navigation bar with links: HOME, DOCS, SUPPORT, ACCOUNT, and LOGOUT. Below this, the 'Development Environment' section is active, with tabs for HOME, METRICS, INFORMATION, and GATEWAY API. The main content area displays various system metrics and a table of nodes. The metrics section shows 3 nodes, a block height of 278103, a protocol of Geth/PoA, a block period of 5 seconds, a chain ID of 590895334, and a release version of 1.0.21. The nodes table lists three nodes: Node 1, Node 2, and System Monitor, all with a status of 'started'. Below the nodes table, there's a 'SERVICES' section listing Block Explorer, Ether Pool, and Token Explorer.

Figura 4.18: Interfaz de la consola de Kaleido donde se encuentran los nodos creados del entorno.

En nuestro caso, como hemos usado Kaleido junto a Ethereum, hemos usado un generador web (Mnemonic Code Converter) que es el que nos ha generado las distintas series de palabras mnemotécnicas para establecer la conexión, así como las direcciones de las 10 cuentas con 100 de ETH con sus respectivas claves pública y privada. [18]

Dentro de la web Kaleido, una vez ya creamos la red (denominada consorcio), el entorno y el nodo, tuvimos que enlazar este con su respectiva wallet con fondos. Finalmente, el último paso fue enlazar este nodo, mediante sus credenciales de conexión, a nuestro archivo de configuración de proyecto en Sublime Text. [19]

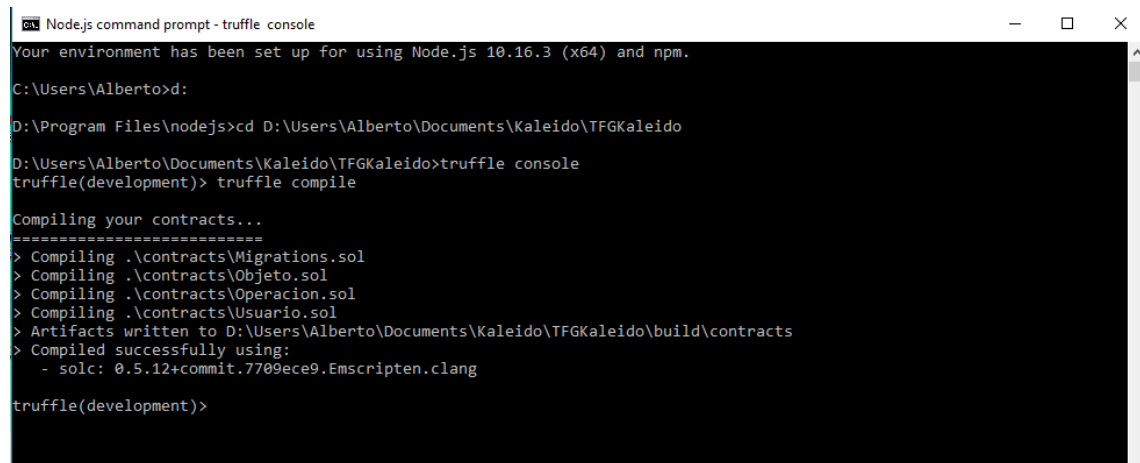


4.19: Interfaz de la consola de Kaleido con las credenciales necesarias para la conexión.

```
networks: {
  development: {
    provider: () => {
      const appCred = 'e0vth0j197:a0xmH0yiW2CDacPOxVZZEdPeIh9GhLJA1JyLWc-P7kM'; // from application credential widget
      const connectionURL = 'e0x7h51p68-e0jmqb4z85-rpc.de0-aws.kaleido.io'; // without protocol (https://)
      return new HTTPProviderRateLimitRetry(`https://${appCred}@${connectionURL}`, 100000);
    },
    network_id: "*", // Match any network id
    gasPrice: 0,
    gas: 4500000,
    /* type: 'quorum' // Use this property for Quorum environments */
  },
}
```

Figura 4.20: Estructura del proyecto: Archivo de configuración de Kaleido en Sublime Text.

Finalmente hay que añadir que, para poder ejecutar el proyecto, es necesario el uso de la consola de comandos de Node.js. Además de esto, también la usamos para acceder a la consola de Truffle, encargada de la compilación y migración de los distintos contratos del proyecto.



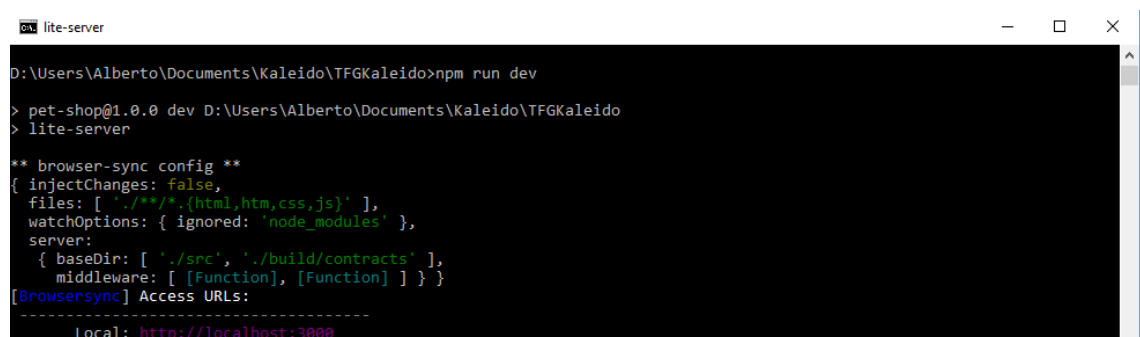
```
Node.js command prompt - truffle console
Your environment has been set up for using Node.js 10.16.3 (x64) and npm.

C:\Users\Alberto>d:
D:\Program Files\nodejs>cd D:\Users\Alberto\Documents\Kaleido\TFGKaleido
D:\Users\Alberto\Documents\Kaleido\TFGKaleido>truffle console
truffle(development)> truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\Objeto.sol
> Compiling .\contracts\Operacion.sol
> Compiling .\contracts\Usuario.sol
> Artifacts written to D:\Users\Alberto\Documents\Kaleido\TFGKaleido\build\contracts
> Compiled successfully using:
   - solc: 0.5.12+commit.7709ece9.Emscripten.clang

truffle(development)>
```

Figura 4.21: Consola de Truffle.



```
lite-server
D:\Users\Alberto\Documents\Kaleido\TFGKaleido>npm run dev

> pet-shop@1.0.0 dev D:\Users\Alberto\Documents\Kaleido\TFGKaleido
> lite-server

** browser-sync config **
{ injectChanges: false,
  files: [ './**/*.html', './**/*.css', './**/*.js' ],
  watchOptions: { ignored: 'node_modules' },
  server:
    { baseDir: [ './src', './build/contracts' ],
      middleware: [ [Function], [Function] ] } }
[Browsersync] Access URLs:
-----
Local: http://localhost:3000
```

Figura 4.22: Ejecución del proyecto Kaleido dentro de la consola de Node.js.

5

Manual de Usuario

Lo que pretendemos crear es un sistema de almacenamiento que permita ir guardando todos los registros que se lleven a cabo en el ciclo vida del animal, desde el inicio donde el animal nace en la granja (donde se debe almacenar los datos sobre el animal nacido), pasando por las fases de inspecciones de sanidad que se realicen sobre los animales (que deberán también estar guardadas), del transporte de dicho animales a los distintos lugares (al matadero, otra granja), del propio matadero y también tendremos que registrar los procesos de alimentación de los distintos animales.

Para empezar a hablar un poco sobre la web que hemos diseñado, primero tenemos que explicar el sistema de roles que hemos mencionado antes, ya que es necesario aclarar para ver con claridad el funcionamiento de la web creada.

Disponemos de 4 tipos de roles que se encargan de dejar constancia sobre los distintos procesos que influyen en el ciclo de vida de los animales. Estos roles son granjero (encargado de la alimentación), inspector (encargado de la inspección), transportista (encargado del proceso de transporte donde pueden ser llevados varios animales) y matarife (entidad encargada de registrar los datos de los animales cuando son sacrificados en el matadero).

Además de estos roles, disponemos de un rol extra, el Administrador, capaz de poder insertar en los 4 tipos de operaciones disponibles (engloba los 4 roles), de poder añadir animales a la base de datos y será el rol encargado de usar el apartado de simulación, donde realizaremos las pruebas de rendimiento.

La vista y la estructura de la página son similares entre ellas, ya que queríamos tener una página web común, sobre la que adaptar las demás bases de datos. Es por esto por lo que la página web cuenta con los mismos apartados distribuidos de la misma forma, salvo con una modificación del estilo y del diseño de las relacionales y no relacionales con respecto a la Blockchain, pero es únicamente visual. En concreto, la capturas que aparecen en este manual son con el estilo y diseño de las páginas web relacionales y no relacionales que hemos desarrollado.

Lo primero que nos encontramos al acceder a la página web es una ventana de inicio de sesión donde podemos entrar con nuestro usuario rellenando los campos DNI y contraseña y dándole a Entrar.

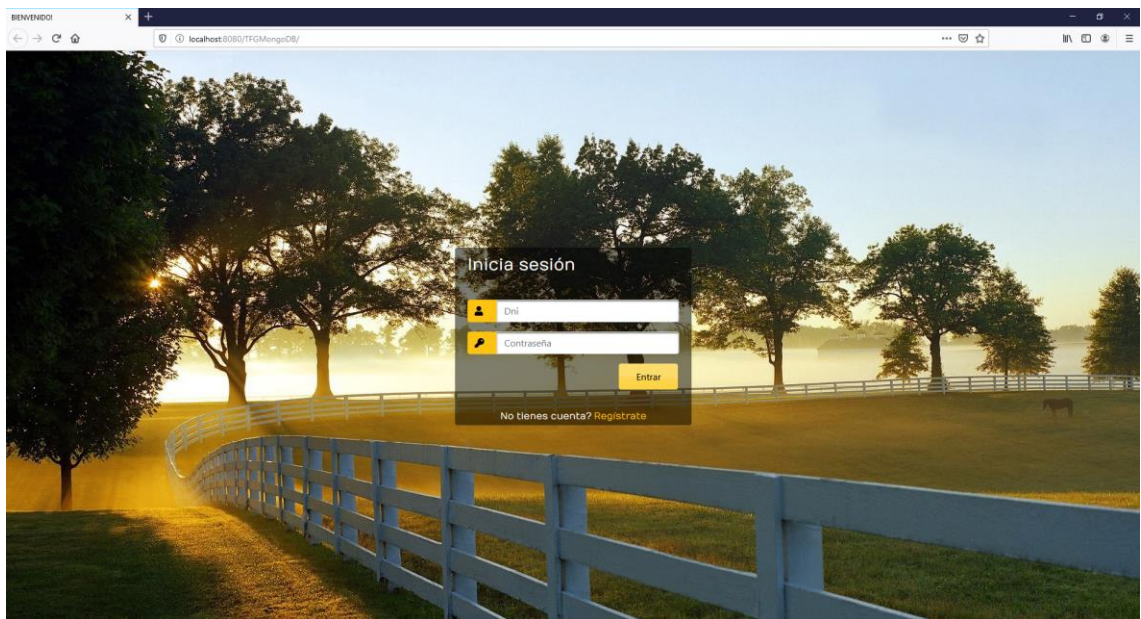


Figura 5.1: Vista de la pantalla de inicio de sesión.

Además, también podemos encontrar un enlace donde deberás acceder si no estás aún registrado. Al pinchar en el enlace, te redirigirá a otra vista, concretamente la del registro, donde deberás introducir la información sobre los distintos campos solicitados: el nombre, los apellidos, el email, el DNI, el rol (esto repercutirá en lo que puedes ver y hacer dentro de la página), la contraseña y la confirmación de esta.

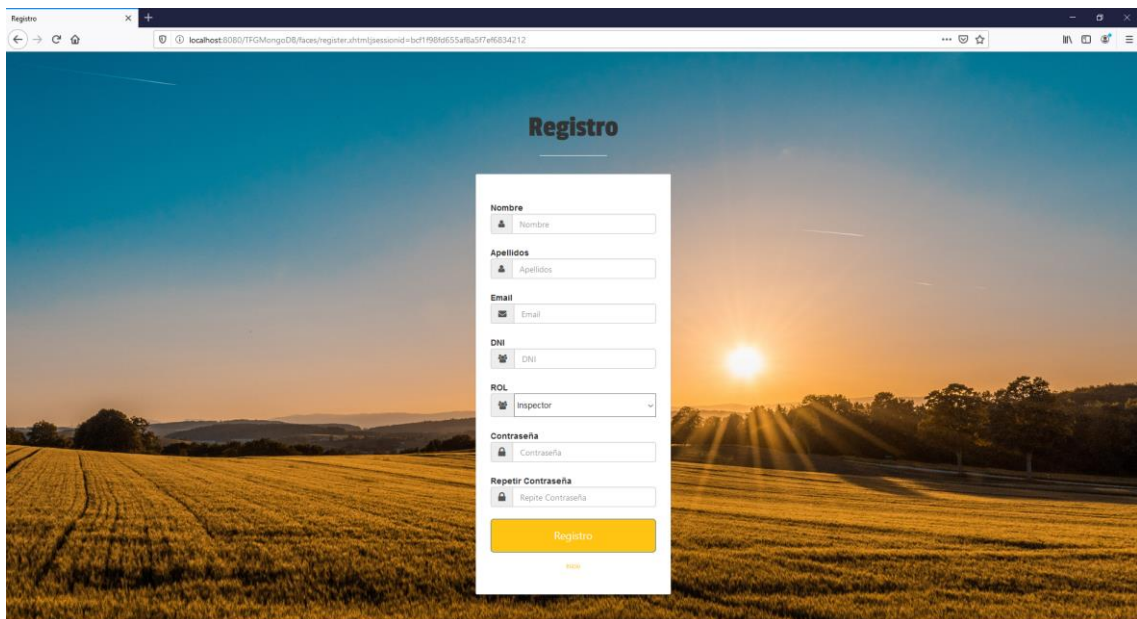


Figura 5.2: Vista de la pantalla para realizar un registro de usuario.

Cuando ya se hallan introducido los datos, ya sea mediante el registro o iniciando sesión, no redirigiremos a la vista de la pantalla de Inicio. La vista que veremos ahora es entrando con un usuario que posee el rol de administrador.

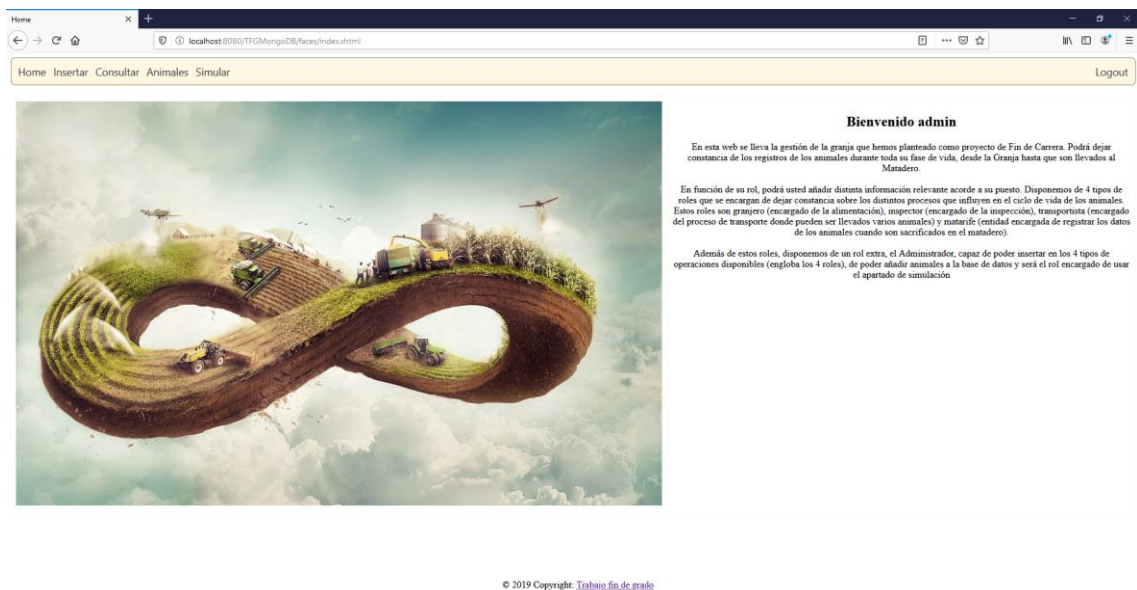
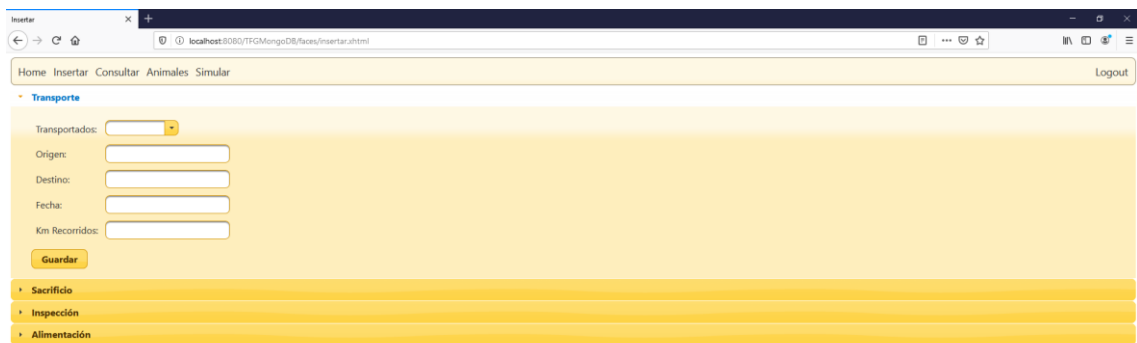


Figura 5.3: Vista de la pantalla del menú de inicio (Home).

Como podemos ver, en este apartado se encuentran varias pestañas: Home (te redirige a la página de Inicio en la que nos encontramos), Insertar (te permite insertar operaciones), Consultar (para consultar operaciones realizadas a los animales), Animales (inserción de animales en la base de datos, solo disponible para el administrador) y Simulación (donde realizaremos las pruebas de rendimiento, solo disponible para el administrador). También se encuentra otra pestaña más que es la que deberemos pulsar cuando queramos desconectarnos, denominada Logout.

Lo siguiente a analizar es la vista Insertar, en esta vista podrás insertar únicamente lo referente a tu rol, es decir, si eres transportista, sólo podrás insertar transportes que realices. Esta vista se adapta en función del rol que seas, controlando de este modo que no se pueda introducir una operación que no esté acorde a su rol. A continuación, mostraremos las distintas inserciones de operación disponibles (en este caso podemos ver las 4 operaciones ya que hemos iniciado sesión con el rol Administrador).

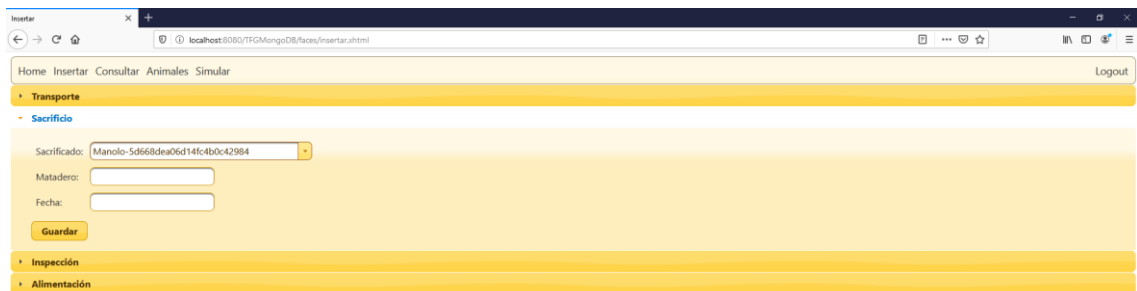
La primera a analizar es Transporte, donde vemos un campo desplegable denominado Transportados (animal/es que vayas a desplazar), Origen (punto del que partes), Destino (punto de llegada al que vas a desplazarte), Fecha (fecha de la operación) y Km Recorridos.



The screenshot shows a web browser window with the URL `localhost:8080/TFGMongoDB/faces/insertar.xhtml`. The page has a navigation bar with links: Home, Insertar, Consultar, Animales, and Simular. A 'Logout' button is in the top right. The main content area is titled 'Transporte' and contains a form with the following fields: 'Transportados' (a dropdown menu), 'Origen' (a text input), 'Destino' (a text input), 'Fecha' (a text input), and 'Km Recorridos' (a text input). Below these fields is a 'Guardar' button. At the bottom of the form, there are three expandable sections: 'Sacrificio', 'Inspección', and 'Alimentación', each with a plus icon.

Figura 5.4: Vista de la pantalla para insertar una operación del tipo Transporte.

La siguiente operación para analizar es la de Sacrificio, donde vemos el campo Sacrificado (animal sacrificado), matadero (nombre del matadero donde se realizó el sacrificio) y Fecha.



The screenshot shows the same web browser window, but the 'Sacrificio' section is expanded. The form fields are: 'Sacrificado' (a dropdown menu showing the value 'Manolo-5d668dea06d14fc4b0c42984'), 'Matadero' (a text input), and 'Fecha' (a text input). A 'Guardar' button is located below these fields. The 'Transporte' section is collapsed, and the 'Inspección' and 'Alimentación' sections remain collapsed at the bottom.

Figura 5.5: Vista de la pantalla para insertar una operación del tipo Sacrificio.

Después nos encontramos con la operación de Inspección, donde encontraremos los campos Inspeccionado (animal al que se le va a realizar la inspección), Descripción (toda información relevante sobre la inspección realizada) y Fecha.

The screenshot shows a web browser window with the URL `localhost:8080/TFGMongoDB/faces/insertar.xhtml`. The page has a navigation bar with links: Home, Insertar, Consultar, Animales, and Simular. A 'Logout' button is in the top right. Below the navigation bar, there are four expandable sections: 'Transporte', 'Sacrificio', 'Inspección' (which is currently expanded), and 'Alimentación'. The 'Inspección' section contains a form with the following fields:

- 'Inspeccionado:' with a dropdown menu showing 'Manolo-5d668dea06d14fc4b0c42984'.
- 'Descripción:' with a text input field.
- 'Fecha:' with a date input field.
- A 'Guardar' button at the bottom.

Figura 5.6: Vista de la pantalla para insertar una operación del tipo Inspección.

Por último, la operación de Alimentación, que posee los campos Alimentado (animal alimentado), Producto (información relevante sobre el producto usado para alimentar al animal) y Fecha.

The screenshot shows the same web browser window as Figure 5.6, but the 'Alimentación' section is now expanded. The form fields are:

- 'Alimentado:' with a dropdown menu showing 'Pakito-5d668bc106d14fc4b0c42982'.
- 'Producto:' with a text input field.
- 'Fecha:' with a date input field.
- A 'Guardar' button at the bottom.

Figura 5.7: Vista de la pantalla para insertar una operación del tipo Alimentación.

A continuación, pasamos a hablar sobre la vista consulta, donde vemos 4 tablas distintas en función del tipo de la operación. En caso del usuario pertenecer a los roles granjero, matarife, inspector o transportistas, en esta vista sólo vería la información relacionada a sus operaciones insertadas. Como hemos accedido con el usuario administrador, este puede ver todos los tipos de operaciones de los distintos usuarios.

Para este ejemplo hemos insertado una operación de cada tipo para ver reflejado en la tabla como se estructura la información de los datos.

En lo referente a los Transportes, vemos que estos se dividen en los campos descritos anteriormente. En concreto, tenemos un campo extra (el primero), denominado id, que este es el identificador de cada operación. Se trata de un enlace a otra vista que contiene, para este caso, la información de los animales transportados, ya que esta operación es la única en la que se deja constancia con varios animales.

Id	Origen	Destino	Fecha	Recorrido (km)	Transportista (dni)
5e2c19123d287ad563902043	Málaga	Aguilar de la Frontera	01/25/2020	112	admin

Figura 5.8: Vista de la pantalla para consultar las operaciones del tipo Transporte.

Si seleccionamos la id, nos lleva a la siguiente vista, donde se ve información sobre los animales transportados:

ID	Nombre	Tipo
5e2c17fdee67cbb377c124e8	Vaca Lechera	Vaca
5e2c1884ee67cbb377c124e9	Cerdo Ibérico	Cerdo

Figura 5.9: Vista de la pantalla para ver los animales que han sido transportados.

Esta vista es especial para esta operación, donde se muestran de nuevo el id de los animales transportados, el nombre y el tipo del animal. A continuación, si seleccionamos la id del animal, pasaremos a una vista que será común a las demás operaciones ya que esta contendrá información sobre las características de los animales.

Por ejemplo, si seleccionamos el primer animal de nombre Vaca Lechera, aparecerán sus respectivas características, que en este caso vemos como hemos registrado su sexo y su fecha de nacimiento.

Característica	Valor
fecha nacimiento	25/01/2020
SEXO	femenino

Figura 5.10: Vista de la pantalla que muestra las características del animal seleccionado.

En referente a la operación de sacrificio, vemos el id de la operación, el matadero, la fecha, el id del animal (enlace que te lleva a las características del mismo), el nombre, el tipo y el DNI del usuario que haya hecho la inserción que en este caso ha sido el usuario Administrador que tiene como DNI admin.

Id	Matadero	Fecha	Animal ID	Nombre	Tipo	Matarife (dni)
Se2c196d3d287a4563902045	Matadero Los Ordos	01/27/2020	Se2c18cae67cbb377c124ea	Gallo Manolo	Gallo	admin

Figura 5.11: Vista de la pantalla para consultar las operaciones del tipo Sacrificio.

La única diferencia entre la operación de sacrificio de la operación de inspección y de alimentación es un campo referente a cada operación: la descripción del resultado de la inspección y el producto usado para alimentar al animal, respectivamente.

Id	Descripcion	Fecha	Animal id	Nombre	Tipo	Inspector (dni)
Se2c18443d287a4563902042	La vaca se encuentra en muy buen estado, ha pasado todas las pruebas correctamente.	01/25/2020	Se2c17fdee67cbb377c124e8	Vaca Lechera	Vaca	admin

Figura 5.12: Vista de la pantalla para consultar las operaciones del tipo Inspección.

Id	Producto	Fecha	Animal id	Nombre	Tipo	Granjero (dni)
Se2c195a3d287a4563902044	Pienseo rico en proteínas de la marca Sanity	01/25/2020	Se2c18cae67cbb377c124ea	Gallo Manolo	Gallo	admin

Figura 5.13: Vista de la pantalla para consultar las operaciones del tipo Alimentación.

Otra de las vistas que juega un papel esencial en la web es la vista que se encarga de introducir a los animales en la base de datos. Se trata de la vista Animales, donde únicamente el administrador puede insertar estos nuevos animales.

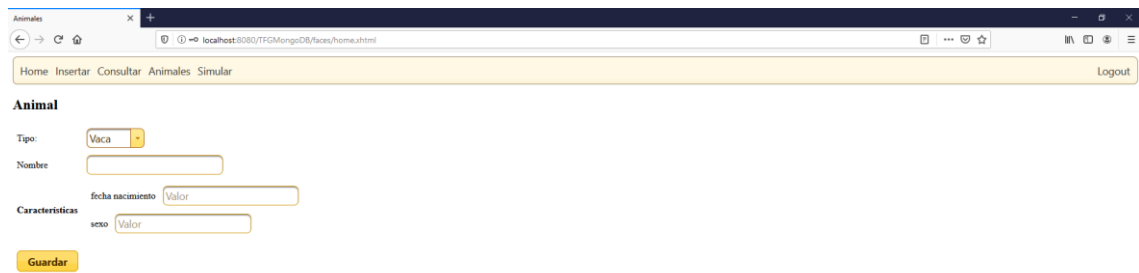


Figura 5.14: Vista de la pantalla dinámica usada para la inserción de los nuevos animales.

En esta vista aparecen los campos tipo y nombre de animal, junto a otro apartado denominado características. En función del tipo de animal seleccionado, el apartado características cambiarán, ajustándose de este modo a su respectivo tipo.

Por último, pero no por ello menos importante, nos encontramos con la pestaña Simular, que básicamente es el núcleo de nuestro TFG ya que las conclusiones las sacaremos en función de los resultados obtenidos en esta vista.

Para esta pantalla necesitaremos especificar el número de inserciones y el número de consultas que deseemos general. Acto seguido, se actualizará el campo de abajo donde pone Tiempo simulación (ms) con el tiempo obtenido de haber generado dichas operaciones en milisegundos.

Cabe destacar, que tanto estas consultas como estas inserciones se generan de forma totalmente aleatoria, y en función del tipo de web que estemos usando, unas tardarán más y otras menos es por esto por lo que, como ya hemos dicho anteriormente, este es el punto clave de nuestro TFG.

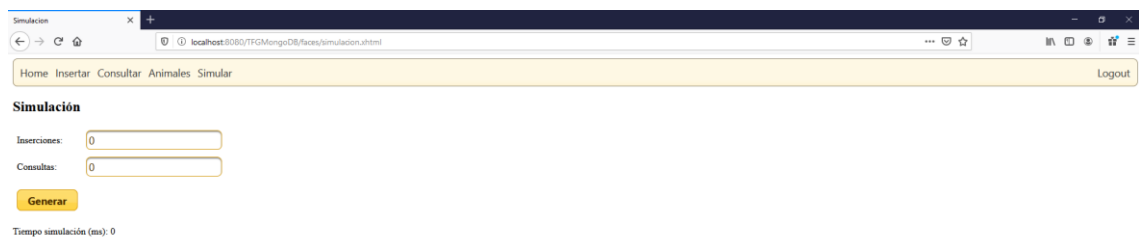


Figura 5.15: Vista de la pantalla donde se insertan el número de inserciones y consultas para realizar la simulación.

6

Resultados

Las pruebas de rendimiento realizadas tanto para PostgreSQL, como para MongoDB y Kaleido están divididas en tres sectores: las inserciones, las consultas y la ejecución de ambas.

En cada proceso, se han realizado pruebas de rendimiento para las cifras de 100, 1000 y 10000 número de operaciones realizadas, ya que estas tres cifras nos resultaban bastante significativas para poder llegar a nuestra conclusión. También hay que añadir que los tiempos que aparecen en las distintas gráficas están en milisegundos.

6.1 Pruebas de rendimiento

6.1.1 PostgreSQL

Con respecto al comportamiento de PostgreSQL en general, cabe decir que este nos ha ofrecido unos buenos tiempos para todas las operaciones, cosa que nos ha sorprendido gratamente.

Como se mostrará en las siguientes capturas, el crecimiento de los tiempos es bastante considerable al pasar de 1000 operaciones a 10000 operaciones, sobre todo en el caso de las inserciones.

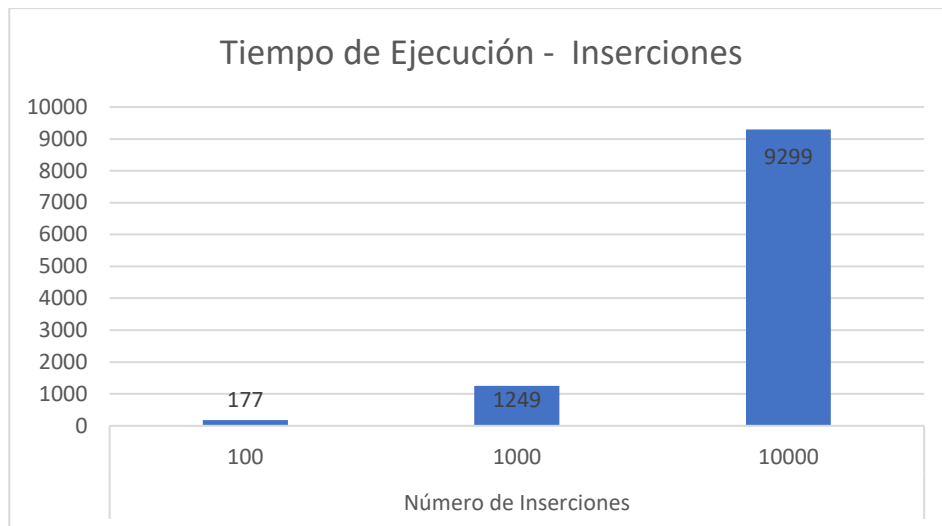


Figura 6.1: Tiempo de ejecución (ms) para las inserciones en PostgreSQL.

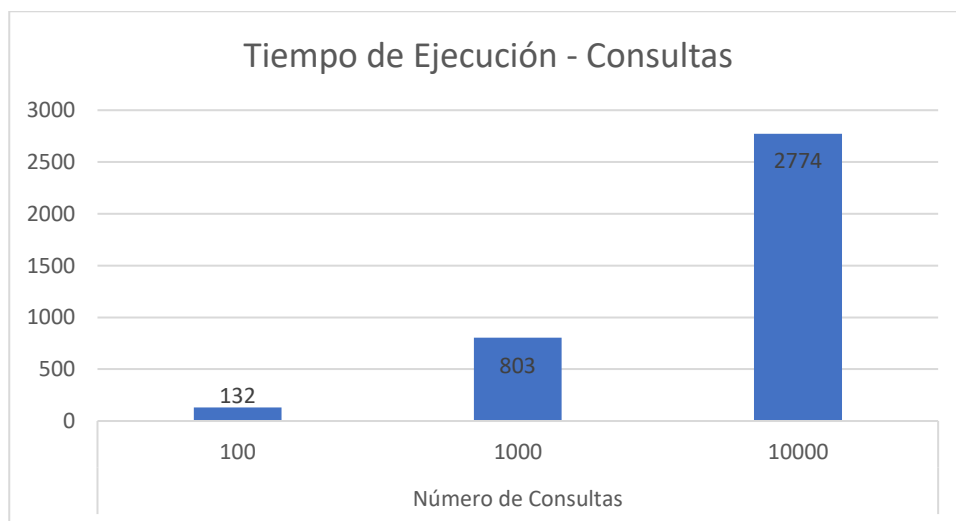


Figura 6.2: Tiempo de ejecución (ms) para las consultas en PostgreSQL.

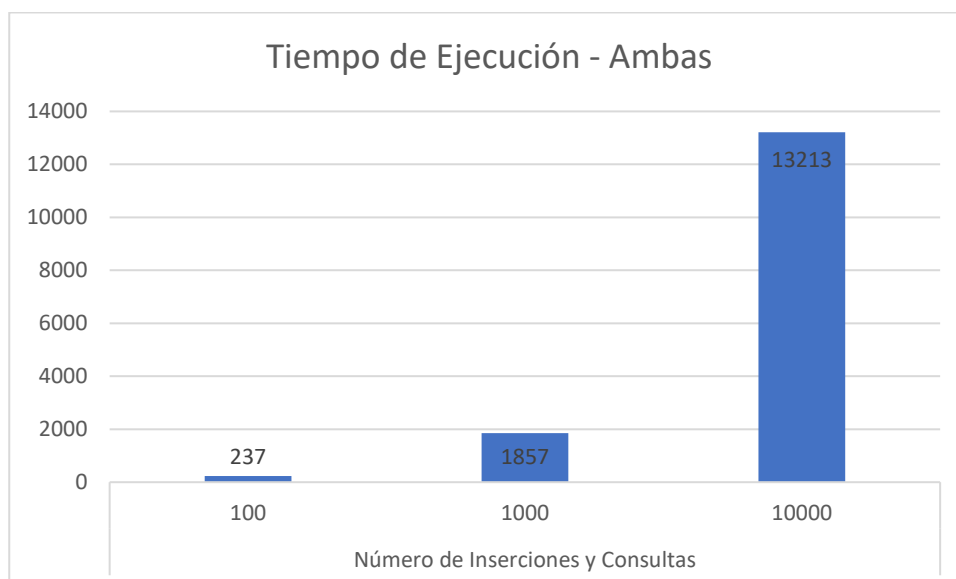


Figura 6.3: Tiempo de ejecución (ms) para las inserciones y las consultas en PostgreSQL.

6.1.2 MongoDB

Con respecto a MongoDB, este se comporta de manera similar a PostgreSQL con las inserciones, salvo que esta tarda un poco más para grandes cantidades de inserciones.

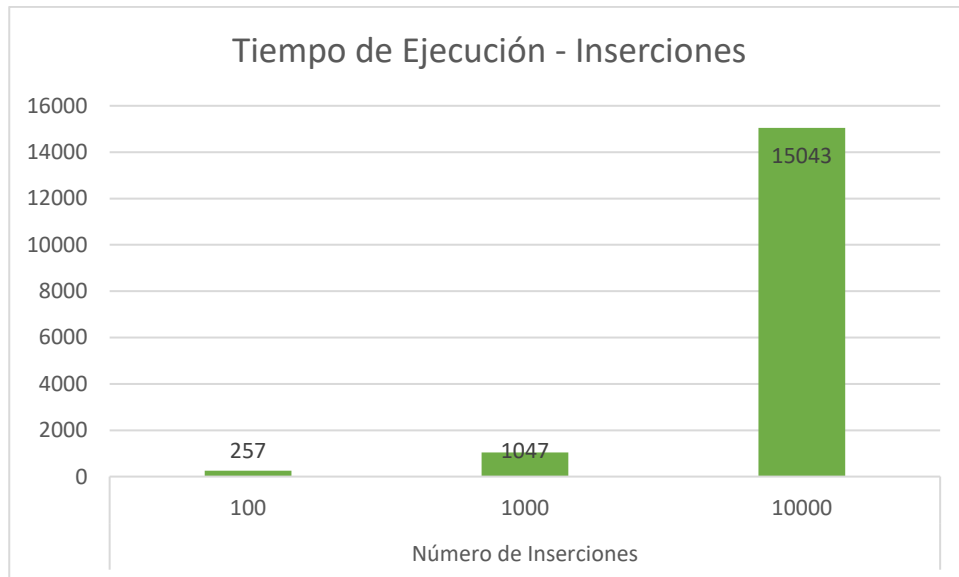


Figura 6.4: Tiempo de ejecución (ms) para las inserciones en MongoDB.

En el apartado de las consultas, MongoDB es ideal para este tipo de operaciones, tal y como podemos comprobar en la gráfica de la figura 6.5.

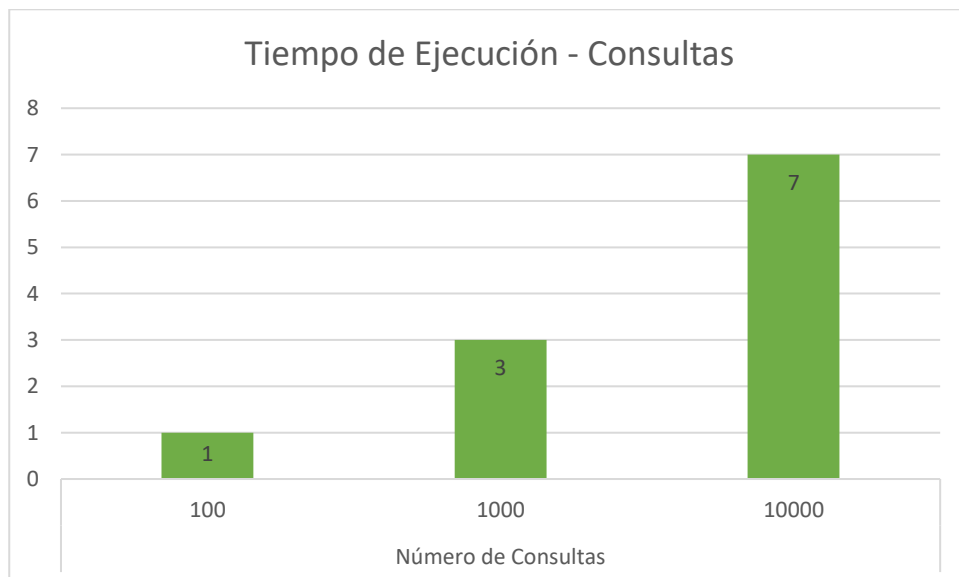


Figura 6.5: Tiempo de ejecución (ms) para las consultas en MongoDB.

Finalmente, en el apartado de ambas operaciones, pese que los tiempos de las consultas son apenas apreciables ya que el mayor valor ha sido 7 milisegundos, en el resultado final salen tiempos un poco mayores de los esperados, pero entra dentro del margen de error debido a la aleatoriedad.

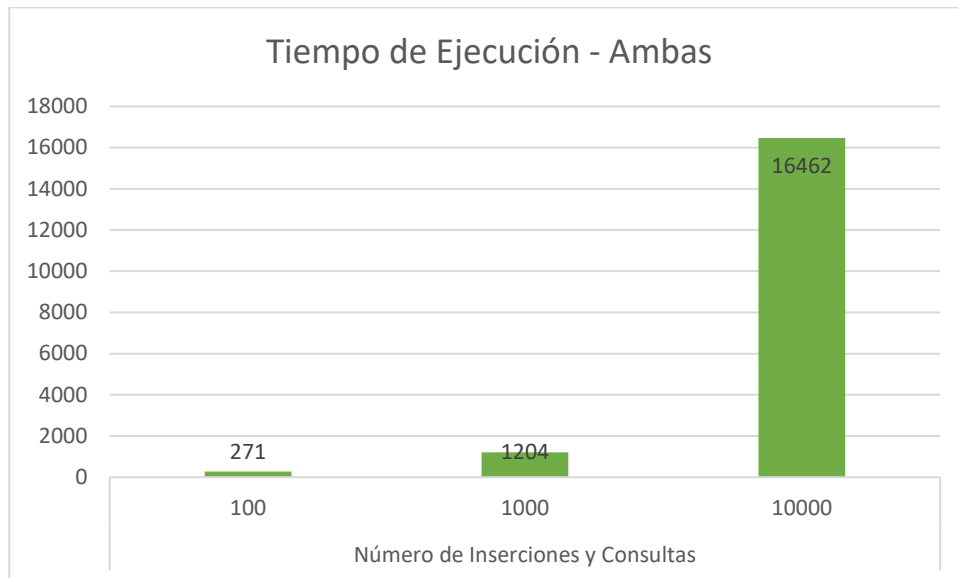


Figura 6.6: Tiempo de ejecución (ms) para las inserciones y las consultas en MongoDB.

6.1.3 Kaleido

Con respecto a Kaleido, los tiempos de ejecución son mucho más mayores de los otros dos tipos de bases de datos, llegando a dar tiempos muy altos para las 10000 operaciones.

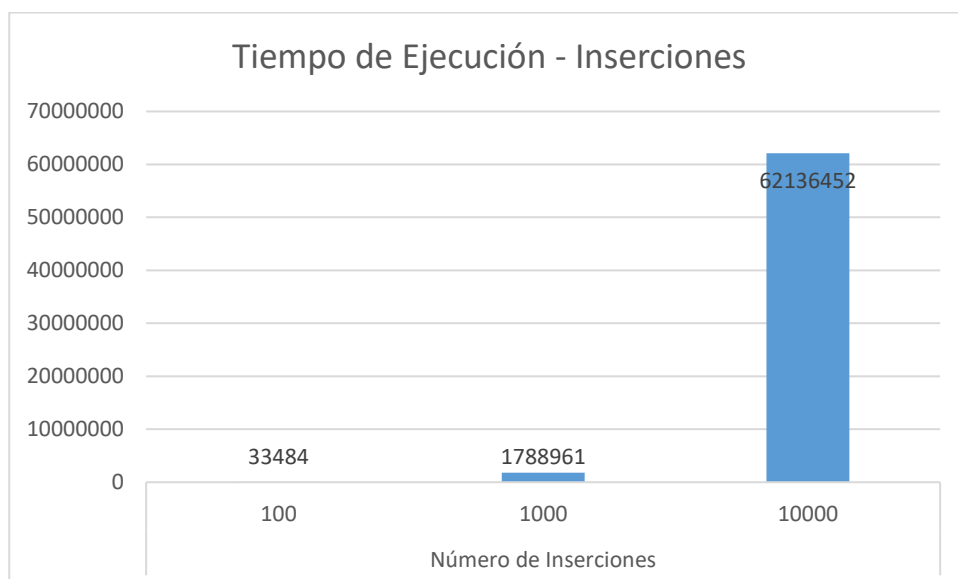


Figura 6.7: Tiempo de ejecución (ms) para las inserciones en Kaleido.

Con respecto a las consultas, el crecimiento del número de operaciones con respecto al tiempo no ha influido de la misma forma en el tiempo de ejecución como en el caso de las inserciones.

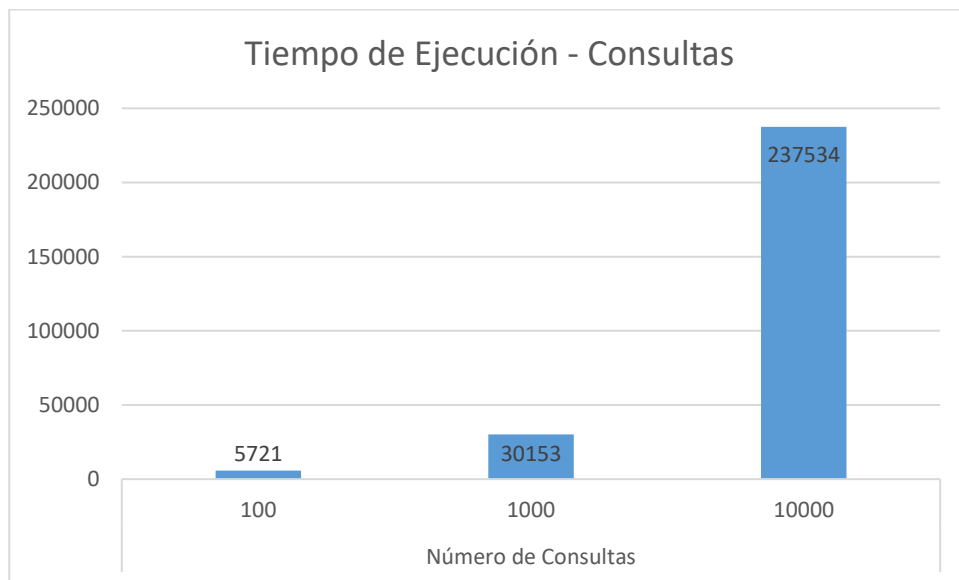


Figura 6.8: Tiempo de ejecución (ms) para las consultas en Kaleido.

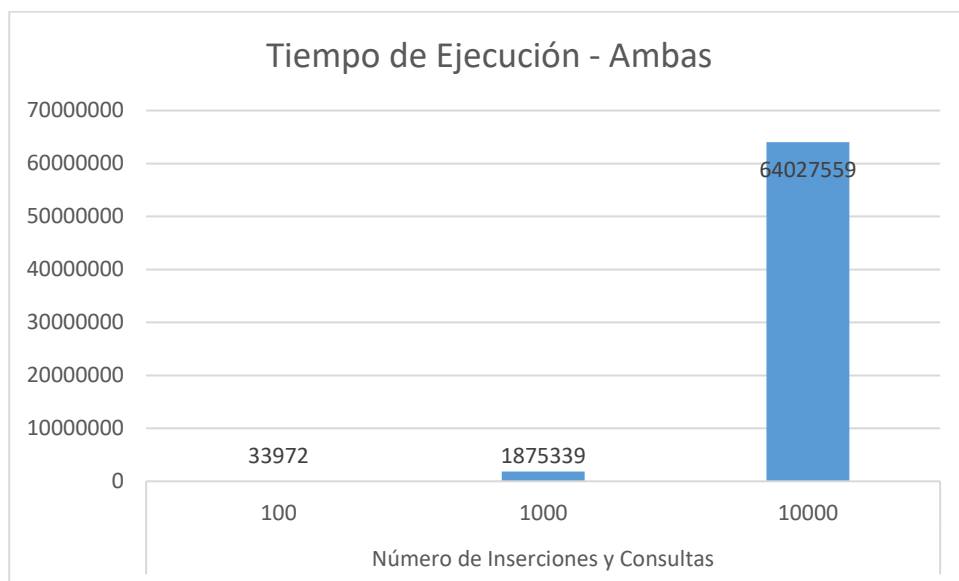


Figura 6.9: Tiempo de ejecución (ms) para las inserciones y consultas en Kaleido.

Como vemos, los tiempos obtenidos son absurdamente altos, llegando a tener que esperar la friolera cantidad de casi 18 horas sin tocar el ordenador para que este completase su ejecución.

6.2 Dificultades

Con respecto a PostgreSQL, que fue nuestra primera elección de bases de datos, tuvimos bastantes problemas sobre todo con respecto a la implementación ya que no sabíamos muy bien por dónde empezar y teníamos que descubrir cómo realizar las distintas conexiones con la base de datos y su respectivo mapeado.

También tuvimos dificultad a la hora de modelar cuál iba a ser nuestra base de datos inicial, ya que las demás parten de la idea de esta pero no fue nada comparado con el tiempo que tardamos en elaborar esta primera aplicación.

Con respecto a MongoDB, tuvimos problemas a la hora de descargar e importar los distintos recursos necesarios para su instalación, así como los drivers necesarios para Netbeans, que se encontraban bastantes escondidos. Sin embargo, el resto de la implementación se hizo más llevadera y no nos demoró tanto tiempo.

Con respecto a Kaleido, esta sí que la consideramos un verdadero problema debido a su escasa información necesaria para el aprendizaje, así como la ausencia de ejemplos útiles, la incompatibilidad de recursos y la complejidad de su implementación. Fue sin duda la que más dificultad nos ha costado en todos los aspectos.

Conclusiones

7.1 Conclusión

Hemos podido aplicar diversos conocimientos aprendidos en la carrera como por ejemplo la generación del modelo entidad relación y usar distintas herramientas que hemos aprendido en algunas asignaturas del curso.

Con respecto al grado de dificultad de las tres tecnologías, pensamos que MongoDB sin lugar a dudas es el gestor de almacenamiento que menos dificultades nos ha causado a la hora de implementación. Seguida de esta se encuentra PostgreSQL, que aporta la facilidad de gestión de la base de datos gracias a su interfaz intuitiva y de fácil uso, pero con respecto a la implementación al principio puede resultar un poco tosca, sobre todo si es de las primeras veces que desarrollas una aplicación web de esta índole. Finalmente y sin lugar a dudas, nuestra experiencia nos dice que Kaleido, y en general las tecnologías Blockchain, suponen una gran dificultad, ya no solo por el aprendizaje y la implementación de la misma, sino también porque al tratarse de una tecnología relativamente nueva que empieza a cobrar importancia, la información disponible no es muy variada y los ejemplos rara vez son muy profundos, ya que siempre nos encontrábamos con el mismo ejemplo que explicaba una aplicación de votos que consistía en una página que introducía un voto y te informaba de los votos realizados.

PostgreSQL cuenta con diversas ventajas que la hacen ser una de las bases de datos relacionales más usadas en la actualidad, destacando que es gratuita tanto su instalación como su uso, es multiplataforma, presenta un sistema de alta disponibilidad, implementa la mayor parte de las funcionalidades principales del estándar SQL, incorpora una herramienta gráfica para la fácil administración de las bases de datos de manera intuitiva, es bastante robusta y fiable al cumplir con el protocolo ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad de los datos) y cuenta con un gran soporte y una infinidad de foros y páginas donde consultar.

Como desventajas destaca que está diseñado para sistemas que vayan a contar con un alto volumen de datos por lo que, de no tener tanta cantidad de información, pueda resultar lenta sobre todo en el caso de las consultas.

MongoDB, por su parte, debido a la dinamicidad de los esquemas, permite editar y agregar cualquier información nueva a los documentos por separado, puede ser escalable, llevar a cabo su desarrollo requiere poca inversión de coste en comparación al resto, realiza muchas operaciones por segundo y es ideal para bases de datos en modo lectura.

Por otro lado, este no es capaz de realizar transacciones y carece de los enlaces (Joins) como en el caso de las relacionales, por lo que, si deseamos consultar datos de varias colecciones, debemos realizar más de una consulta.

Con respecto a las tecnologías Blockchain, pese a la gran seguridad que ofrecen debido a la organización de la cadena de bloques y al sistema de hashes que contiene, el resultado final no es tan favorable, ya que la inversión de desarrollo y de aprendizaje es muy elevado.

A continuación, vamos a pasar a comentar los resultados obtenidos con los tiempos. No cabe duda, que Kaleido ofrece unos tiempos de ejecución demasiado altos para este proyecto por lo que esta tecnología es la que peor rendimiento nos aporta. Con respecto a las bases de datos relacionales y las no relacionales, ambos resultados obtenidos son muy similares y si nos tuviéramos que decantar únicamente fijándonos en el tiempo que tarda tanto para inserciones como para consultas, la elección sería PostgreSQL. Si lo que necesitamos es una base de datos para una aplicación donde sólo se realicen consultas, es decir, sea una base de datos documental, la elección sería MongoDB.

Otra cosa a tener en cuenta es el número de operaciones a realizar. Como vemos en la gráfica de la figura 7.1, si vamos a realizar un número reducido de operaciones en torno a las 1000 y 5000 operaciones, nos decantaríamos por MongoDB, ya que los tiempos en las inserciones son similares y no puede hacerles frente a las lecturas.

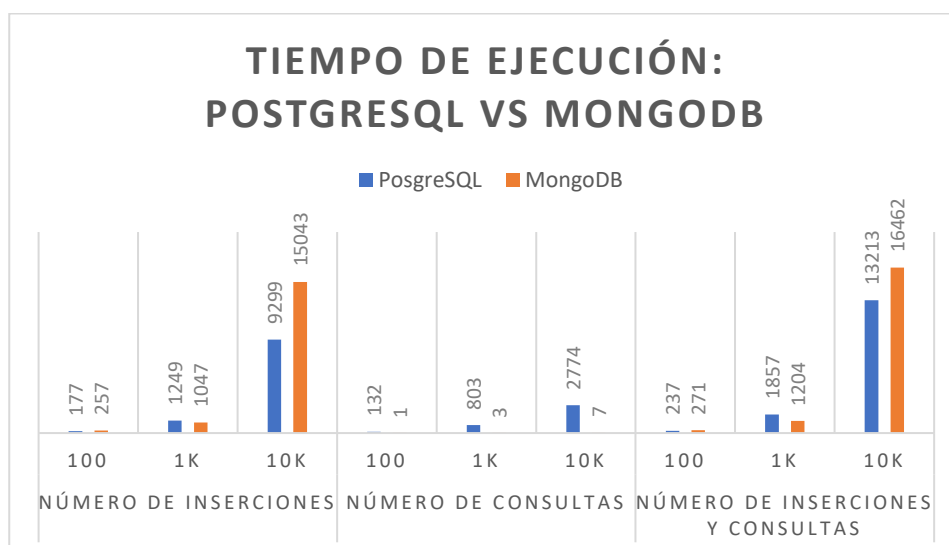


Figura 7.1: Gráfica con los tiempos de ejecución (ms) de PostgreSQL frente a MongoDB

7.2 Opinión sobre el trabajo

Antes de realizar el proyecto, únicamente conocíamos las bases de datos relacionales, pero nunca habíamos trabajado con PostgreSQL ni habíamos elaborado una base de datos desde cero, ni mucho menos habíamos conectado una aplicación web a una base de datos, por lo que en general, los conocimientos eran bastantes reducidos.

Es por esto una de las razones por las que decidimos realizar este TFG, ya no sólo por la novedad de las cadenas de bloques, sino por el conocimiento que íbamos a conseguir tras realizarlo.

Es por esto que hemos aprendido bastante sobre estas tres tecnologías y hemos conocido cómo funcionan internamente, así como el aprendizaje de nuevas herramientas y nuevos lenguajes de programación, cosa que influye positivamente en nuestra trayectoria como ingeniero.

En lo relativo al contenido de este proyecto, ha sido muy interesante conocer el punto de vista de cada tecnología y ver los pros y contras de cada una de ellas, sobre todo para adquirir el conocimiento necesario para saber decidir qué tipo de base de datos necesito para cada caso específico. También ha sido bastante reconfortante aprender y finalizar la implementación de la aplicación con Blockchain, ya que, pese a la escasez de información por las distintas implementaciones y el gran tiempo invertido, el resultado obtenido ha sido bastante favorable y estamos muy contentos con la labor conseguida.

Bibliografía

- [1] https://es.wikipedia.org/wiki/Base_de_datos
- [2] <https://www.grapheverywhere.com/tipos-bases-de-datos-clasificacion/>
- [3] <https://www.tecnologias-informacion.com/basesdedatos.html>
- [4] <https://searchdatacenter.techtarget.com/es/definicion/SQL-o-lenguaje-de-consultas-estructuradas>
- [5] <https://aws.amazon.com/es/nosql/>
- [6] <https://medium.com/@marlonmanzo/sql-vs-nosql-ventajas-y-desventajas-849ccc9db3d4>
- [7] <https://blocktelegaph.io/blockchain-before-bitcoin-history/>
- [8] <https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/>
- [9] <https://www2.deloitte.com/us/en/insights/topics/understanding-blockchain-potential/global-blockchain-survey.html>
- [10] <https://www.investopedia.com/terms/s/silk-road.asp>
- [11] <https://blog.infranetworking.com/servidor-postgresql/>
- [12] https://www.postgresql.org/support/professional_support_europe/
- [13] <https://aws.amazon.com/marketplace/pp/ConsenSys-Kaleido-Enterprise-Blockchain-SaaS/B07CSLDS7R>
- [14] <https://www.ledgerinsights.com/kaleido-enterprise-ethereum-marketplace/>
- [15] <https://azuremarketplace.microsoft.com/en-us/marketplace/apps/consensus.kaleido?tab=Overview>
- [16] <https://academy.bit2me.com/que-son-los-smart-contracts/>
- [17] <https://solidity-es.readthedocs.io/es/latest/>
- [18] <https://medium.com/coinmonks/connecting-to-kaleido-private-ethereum-blockchain-with-truffle-7ae7cc6a3234>
- [19] <https://docs.kaleido.io/kaleido-platform/foundational-concepts/kaleido-key-terms>



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga